VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zastínění okolím
Ambient Occlusion

2014

Bc. Martin Kohoutek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Martin Kohoutek**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Zastínění okolím
Ambient Occlusion

Zásady pro vypracování:

Cílem práce je detailně popsat a následně implementovat techniku globálního osvětlení s využitím OpenGL pro zobrazení modelů v reálném čase. Implementaci proveďte v jazyce C++ za dodržení Google C++ Style Guide a kód důkladně zdokumentujte pomocí nástroje Doxygen.

1. Seznamte se s technikami globálního osvětlení, a to zejména s SSAO, HBAO a HSAO.
2. Metody vzájemně porovnejte a vybranou techniku detailně popište.
3. Proveďte implementaci v C++ využívající OpenGL verze 4.x a odpovídající GLSL.
4. Funkčnost demonstrujte minimálně na tzv. Cornell Boxu a vybraných modelech z [3].

Seznam doporučené odborné literatury:

[1] Bunnell, M. Dynamic Ambient Occlusion and Indirect Lighting. In GPU Gems 2. pp. 223–233. 2005
[2] Hoberock, J., Jia, Y. High-Quality Ambient Occlusion. In GPU Gems 3. 2008.
[3] The Stanford 3D Scanning Repository

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Fabián**

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych poděkoval Ing. Tomáši Fabiánovi za umožnění práce na tomto tématu a podpoření mého zájmu o oblast počítačové grafiky. Také děkuji za návrhy a připomínky k práci.

Abstrakt

Tématem diplomové práce je pochopení a následná implementace nejpoužívanějších algoritmů, které slouží k výpočtu zastínění okolím ve 3D scéně. Zastínění bodu scény okolím udává množství světla dopadajícího do daného bodu, což způsobuje zatemnění rohů objektů a kontaktní stíny mezi objekty. Toto zastínění je dílčím jevem globálního osvětlení. Proto je první část práce zaměřena na metody, které počítají globální osvětlení nebo jeho části. Ve druhé části práce se věnuji přímo zastínění okolím. Počínaje zmíněním první metody počítající zastínění v každém bodě 3D scény, až po nejpoužívanější aproximace tohoto jevu, které pracují nad informacemi o pixelech obrazu získanými během vykreslování.

Klíčová slova: zastínění okolím, globální osvětlení, vykreslení v reálném čase, aproximace v prostoru obrazu

Abstract

The topic of this diploma thesis is understanding and subsequent implementation of the most widely used algorithms that are used to calculate ambient occlusion in a 3D scene. Occlusion of a point in scene indicates the amount of light reaching to that point, which causes darkening the corners of objects and contact shadows between objects. This occlusion is a partial phenomenon of global illumination. Therefore, the first part of this thesis is focused on methods that calculate global illumination or its part. The second part is devoted directly to ambient occlusion. Starting by mentioning the first method that calculates occlusion at each point in a 3D scene, to the most commonly used approximation of this phenomenon, which operate over information about pixels of image acquired during rendering.

Keywords: ambient occlusion, global illumination, real-time rendering, screen space approximation

Seznam použitých zkratek a symbolů

BRDF	–	Bidirectional Reflectance Distribution Function
HDR	–	High Dynamic Range
fps	–	Frames Per Second
AO	–	Ambient Occlusion
SSAO	–	Screen Space Ambient Occlusion
HSAO	–	Horizon Split Ambient Occlusion
HBAO	–	Horizon Based Ambient Occlusion
SAO	–	Scalable Ambient Obscurance
MVAO	–	Multiview Ambient Occlusion
MIP	–	Multum In Parvo
OpenGL	–	Open Graphics Library
GLSL	–	OpenGL Shading Language

Obsah

Seznam tabulek

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Herní průmysl se neustále vyvíjí a vývojáři se snaží své aplikace co nejvíce přiblížit realitě, a to hlavně po grafické stránce s využitím metod jako jsou měkké stíny, HDR a další post-process efekty. Vylepšují se detaily modelů, zvyšuje se celkový počet objektů ve scéně a zdokonaluje se výpočet osvětlení scény. Právě osvětlení hraje velkou roli ve výsledném snímku aplikace. Pro výpočet přesnějšího a přesvědčivějšího osvětlení ve scéně slouží metody globálního osvětlení.

Globální osvětlení vytváří zastínění scény pod objekty a kolem nich (tzv. kontaktní stíny), krvácení barev, kaustiky a další. Tyto metody určují osvětlení nejen z příspěvků paprsků světelného zdroje (přímé osvětlení), ale také z odražených paprsků od objektů scény (nepřímé osvětlení). Mezi hlavní metody globálního osvětlení patří ray tracing, path tracing, sledování fotonů, radiozita. Efekty globálního osvětlení jsou velice jemné a jako samotné se mohou zdát nepodstatné, ale ve výsledném obraze scény jsou hlavními prvky přidávajícími na realističnosti. Tyto metody jsou ovšem výpočetně náročné a pro vykreslování herní grafiky v reálném čase zatím nepoužitelné. Proto bylo nutné upustit od určitých aspektů globálního osvětlení a vznikl ambient occlusion (zastínění okolím), který aproximuje osvětlení prostředí.

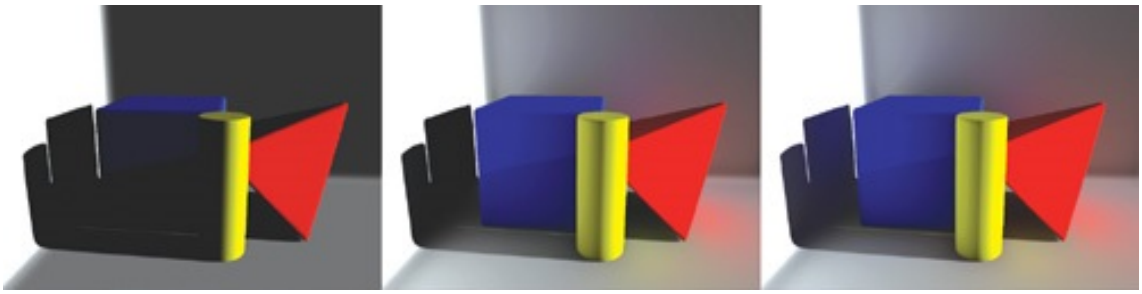
Ambient occlusion je algoritmus, který nepočítá všechny prvky globálního osvětlení, ale věnuje se pouze části přímého osvětlení. Je to aproximace velikosti osvětlení daného bodu, který může být zastíněn objekty scény. Přidává tak do obrazu kontaktní stíny mezi objekty, které přidávají dojem hloubky a vzájemné polohy objektů. Výpočet zastínění byl do příchodu screen-space metod prováděn jako přípravný krok před vykreslovací smyčkou. Tento výpočet stačilo provést pouze jednou, kdy se získané hodnoty uložily do textury a poté se používaly při vykreslování. Daný postup je vhodný pro statické scény. Pro dynamické scény by bylo třeba zastínění přepočítávat a proto vznikly právě screen-space algoritmy, které výpočet zastínění aproximují.

Cílem práce bylo seznámit se právě s těmito metodami výpočtu zastínění, porozumět jim a zvolenou metodu naimplementovat. Vybral jsem si metody SSAO (4.1) a HBAO (4.3), které jsou v současné době těmi nejpoužívanějšími.

2 Globální osvětlení

Představte si venkovní scénu za denního svitu, kdy je aktivní pouze jeden zdroj světla - Slunce. V tomto případě jsou objekty vystavené přímému záření osvětleny a jasně viditelné. Ostatní mají mezi sebou a zdrojem světla nějakou překážku a jsou ve stínu. Nemají už tak výraznou barvu, ale nejsou zcela černé. To, že vidíme objekty, které nejsou přímo vystaveny záření a jsou ve stínu znamená, že buď jsou tyto objekty zdrojem a světlo emitují nebo odráží světlo přicházející odjinud. Protože většina objektů světlo neemituje, jedná se o druhý případ.

Světlo se tedy šíří prostorem tak, že se odráží od povrchu objektů. Odráží se do všech směrů s různou intenzitou. Po kontaktu s povrchem se vždy určitá část světla odrazí a část je povrchem absorbována. Do oblastí zakrytých překážkami se tedy světlo šíří z různých směrů. Část příchozího světla se odrazí od atmosféry, která se tímto jeví jako modrá. Slunce avšak poskytuje tolik světla, aby paprsky procházející atmosférou, které se odráží mezi objekty, mohly vytvořit všudypřítomné světlo prostředí, díky kterému prakticky nenarazíme na absolutně černý stín.



Obrázek 1: Vliv nepřímého osvětlení. První obraz je vytvořen jen s pomocí přímého osvětlení. Na dalších je přidáno nepřímé osvětlení 1 odraz a 2 odrazy každého paprsku ve scéně. Převzato z [1]

Tento princip vzájemného osvětlování ploch se nazývá interreflekce (nepřímé osvětlení). Osvětlovací model v počítačové grafice, který řeší vzájemné záření se nazývá globální osvětlení. Představuje paprsky světla odrážející se ve scéně mezi objekty, které nakonec dorazí do oka pozorovatele (kamery). Běžný vykreslovací postup, který počítá pouze se světlem přicházejícím přímo ze světelného zdroje se nazývá přímé osvětlení. Globální osvětlení počítá přímé i nepřímé osvětlení ve scéně, které představuje vykreslovací rovnice:

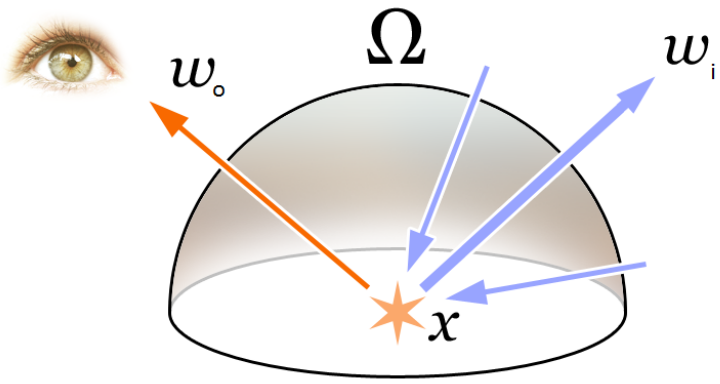
$$L_o(X, \omega_\mathbf{o}) = L_e(X, \omega_\mathbf{o}) + \int_\Omega f_r(X, \omega_\mathbf{i}, \omega_\mathbf{o})\ L_i(X, \omega_\mathbf{i})\ (\omega_\mathbf{i} \cdot \mathbf{n})\, \mathrm{d}\omega_\mathbf{i}\ , \tag{1}$$

kde L_o je odchozí záření bodu X směrem $\omega_\mathbf{o}$, L_e je materiálem emitované záření, $\int_\Omega$ je integrál přes hemisféru kolem bodu X, f_r je funkce odrazivosti materiálu (BRDF) a L_i je příchozí záření ze směru $\omega_\mathbf{i}$ (inverzní vektor dopadajícího záření). Skalární součin $\omega_\mathbf{i} \cdot \mathbf{n}$

zeslabuje hodnotu příchozího záření podle úhlu mezi těmito vektory. Tuto rovnici lze rozdělit na 3 části, kterými jsou:

- Termín L_e , který reprezentuje vlastní záření povrchu.
- Funkce f_r upravující odražený paprsek.
- Termín $L_i\ (\omega_\mathbf{i} \cdot \mathbf{n})$, který představuje množství příchozího (přímého i nepřímého) záření.

Algoritmy pro výpočet zastínění okolím, které jsou tématem této práce lze reprezentovat právě posledním termínem vykreslovací rovnice, avšak pouze pro přímé osvětlení.

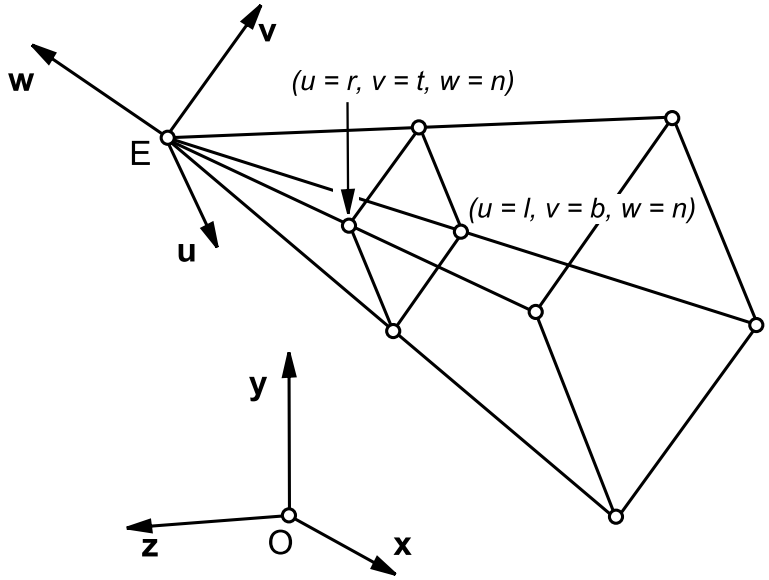


Obrázek 2: Hemisféra kolem bodu X se směrem odchozího záření $\omega_\mathbf{o}$ a inverzním směrem příchozího záření $\omega_\mathbf{i}$. Převzato z [12]

Protože víme, že absolutně tmavé oblasti neexistují a nechceme, aby aplikace vypadaly uměle, je nutností do nich zaintegrovat globální osvětlení, nebo alespoň jeho část. Nyní se podíváme na metody globálního osvětlení, které jsou přesnější a vypadají lépe, ale jsou náročnější na výpočet a v herním průmyslu ještě nepoužitelné.

2.1 Ray tracing (sledování paprsku)

Sledování paprsku je metoda generování obrazu, kdy každým pixelem obrazu vyšleme paprsek a provádíme testování, zda protnul nějakou část geometrie scény. Každý vygenerovaný paprsek shromažďuje při putování scénou informaci o barvě, která bude tvořit výslednou barvu pixelu obrazu. Pro každý paprsek je potřeba projít všechny polygony modelů ve scéně, což je náročné na výpočet. Pro rychlejší vyhledávání se používá stromových struktur, které dělí objekty a jejich trojúhelníky do hierarchické struktury.



Obrázek 3: Souřadný systém (**uvw**) kamery E vůči souřadnému systému scény (**xyz**) a počátku O. Převzato z [2]

Geometrie je zobrazena v **uvw** souřadném systému se středem promítání (pozice kamery) E. Souřadnice w ve výsledku udává vzdálenost od kamery a uv jsou souřadnice pixelu na obrazovce. Máme-li promítací rovinu ve vzdálenosti w = n, potřebujeme vypočítat pozice bodů, které na ni leží, v souřadném systému scény (**xyz**). Tyto body pak tvoří pixely výsledného obrazu. Ze středu promítání E těmito body posíláme paprsky (polopřímky) a provádíme test na protnutí geometrie, která je nejblíže E. Paprsek je definován jako:

$$\mathbf{p}(t) = \mathrm{E} + \mathbf{d}(t) \ , \tag{2}$$

kde E je pozice kamery, **d** je směrový vektor z kamery k pixelu průmětny a t je parametr, který určuje vzdálenost protnutého objektu od kamery. Při kontaktu paprsku s geometrií záleží na vlastnostech povrchu, který ovlivňuje chování paprsku po dopadu a jeho barvu. Informaci o barvě paprsku tvoří ambientní osvětlení objektu, difúzní odraz přímého paprsku světla od povrchu, zrcadlový odraz paprsku světelného zdroje, zrcadlový odraz paprsku přicházejícího od jiného objektu a lomené světlo na průhledném povrchu.

```
vypocitej vektory u, v, w
pro kazdy pixel v obraze
{
  vypocitej paprsek pro dany pixel
  nastav t_min na maximalni hodnotu a vymaz objekt_min

  pro kazdy objekt ve scene proved test na protnuti paprskem
  {
    pokud paprsek protina tento objekt a vzdalenost t je mensi nez t_min
    {
```

```
        t_min = t
        objekt_min = objekt
      }
    }
    pokud nebyl objekt_min nastaven
    {
      nastav barvu pixelu na barvu pozadi
    }
    jinak
    {
      vygeneruj paprsek ke kazdemu zdroji svetla, pro zjisteni stinu
      pokud povrch objektu odrazivy, vygeneruj odrazeny paprsek a rekurzivne testuj paprsek
      pokud je povrch objektu pruhledny, vygeneruj paprsek lomu a rekurzivne testuj paprsek
      vypocitej barvu pixelu pro objekt_min ze ziskanych paprsku
    }
  }
```

Výpis 1: Pseudokód sledování paprsků

V případě doplnění algoritmu o více paprsků v místě průsečíku paprsku s geometrií lze simulovat zastínění scény dané termínem příchozího (přímého) osvětlení ve vykreslovací rovnici 1. S každým takto vygenerovaným paprskem se provede test na protnutí geometrie. Pokud je průsečík vzdálený od počátku paprsku do zvolené hodnoty τ, přispívá tento paprsek k výslednému zastínění bodu.

2.2 Path tracing (sledování cest)

Sledování cest vychází z metody sledování paprsků, avšak je schopno simulovat měkké stíny, kaustiky, zastínění a nepřímé osvětlení, které je v ostatních řešeních nutné speciálně naimplementovat. Tato metoda tedy řeší všechny části vykreslovací rovnice 1.

Vykreslování probíhá tak, že se opět každým pixelem vyšle paprsek. Paprsku se nastaví váha $w = 1$. Při protnutí nejbližšího objektu se v bodě protnutí náhodně rozhodne, zda počítat odražené nebo emitované světlo. Při volbě emitovaného se vrátí hodnota emisní funkce povrchu vynásobená váhou. U reflektovaného paprsku se váha vynásobí hodnotou odrazivosti povrchu a náhodně se vygeneruje odražený paprsek, který se bude rekurzivně trasovat. Takto paprsek pokračuje, dokud nenarazí na zdroj světla. Postup vykreslování může být i opačný, kdy se paprsky generují ze světelných zdrojů a končí na promítací rovině kamery. Takto se vygenerují paprsky pro každý pixel vícekrát a výsledné hodnoty se zprůměrují.

2.3 Radiozita

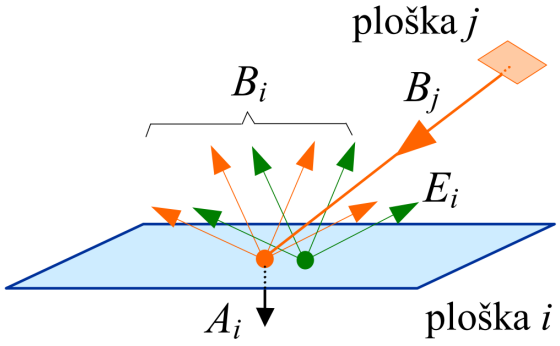
Hlavní využití radiozity je ve výpočtu šíření světla interiérem s difúzními povrchy. Radiozita je množství záření, které opouští plochu. Je závislá na geometrii a materiálech ve scéně a na rozmístění a intenzitě světel. Nezávisí na pozici pozorovatele, proto stačí vyzařování scény vypočítat pouze jednou a poté tyto data využívat k vykreslování. Pro popis algoritmu jsem využil zdroj [3].

Algoritmus počítá se zachováním energie v uzavřeném prostoru, kdy vyzařovaná a absorbovaná energie plošky je rovna energii dopadající na plošku a jejím samotném vyzařování. Na rozdíl od sledování paprsků, kdy každému pixelu obrazu přidělujeme barvu, počítá radiozita intenzitu v každém bodě scény. Pro výpočet intenzit ve scéně je třeba plochy objektů rozdělit na menší. Každé plošce se vypočítá intenzita vyzařování. Protože se předpokládá konstantní vyzařování na cele plošce, je třeba v místech změny intenzity vyzařování síť geometrie zjemnit. Předem ovšem nevíme, jaké vyzařování bude v každém místě scény. Na začátku se hustota sítě nastaví intuitivně. Výpočet se poté provádí ve více krocích, kdy po výpočtu vyzařování se každá síť objektu podle výsledků upraví, dokud není hustota sítí optimální.

Pro výpočet vyzařování plošky slouží tato rovnice:

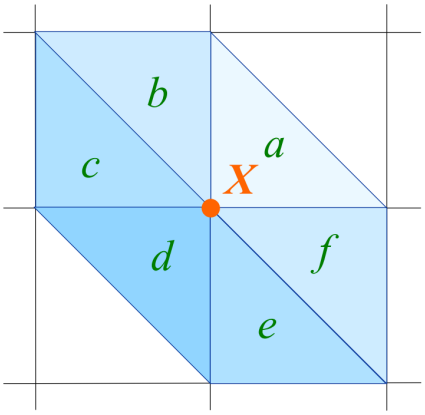
$$B_i = E_i + \rho_i \sum_{j=1}^{n} B_j F_{j \to i} \frac{A_j}{A_i} \ , \tag{3}$$

kde B_i je výsledné vyzařování plošky, E_i je její vlastní záření, ρ_i je koeficient odrazu a n je počet plošek. $F_{j \to i}$ zohledňuje vzájemnou polohu dvou plošek a udává, jak velká část záření plošky j připadne plošce i a A je velikost plošky.



Obrázek 4: Vzájemné záření plošek. Převzato z [3]

Aby při různých intenzitách mezi jednotlivými ploškami nedocházelo ke skokové změně jasu vedoucího k viditelným artefaktům, je třeba mezi hodnotami nějak interpolovat. Nejprve hodnoty vyzařování plošek přeneseme do uzlů sítě. To provedeme zprůměrováním hodnot plošek, kterých je bod sítě součástí. Pro získání intenzit osvětlení na celém povrchu objektu se poté interpoluje mezi hodnotami intenzit ve vrcholech sítě.



Obrázek 5: Vyzařování uzlu jako průměrná hodnota okolních plošek. Převzato z [3]

V tomto případě je intenzita v bodě X rovna součtu plošek a až f, která je podělena jejich počtem. Takto je proveden výpočet pro každý bod uvnitř sítě a jsou zamaskovány velké změny intenzity ozáření mezi sousedními body. Skokové změny intenzit ovšem nechceme zamaskovat na hranách a v rozích objektů, kde bychom ztratili pojem o tvaru objektu. Pro tyto případy se používá extrapolace mezi intenzitami sousedních bodů.

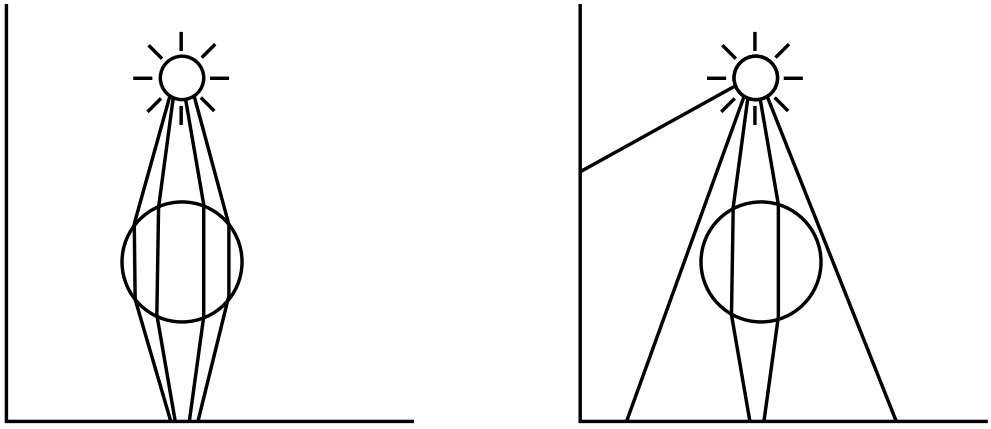
2.4 Fotonové mapy

Fotonové mapy jsou dvou-průchodový algoritmus, který odděluje výpočet osvětlení od geometrie. V prvním kroku se provádí emitování fotonů a jejich ukládání do fotonových map, ve druhém se přistupuje k těmto mapám a provádí se výpočet příchozího a odraženého záření. Pomocí tohoto algoritmu je možné vypočítat nepřímé osvětlení a kaustiky. V porovnání s radiozitou, která je rychlejší pro jednoduché scény s difúzními povrchy, je škálování v komplexních scénách u fotonových map lepší.

Principem algoritmu je podle [4] emitování a sledování fotonů ve scéně. Fotony jsou částice, které jsou emitovány světelnými zdroji ve scéně. Při kolizi s povrchem se uchovávají do fotonových map, které reprezentují množství příchozího ozáření určitého bodu povrchu. Tyto mapy se poté použijí k výpočtu barvy výsledného bodu obrazu. Fotonové mapu jsou nezávislé na pohledu.

Většinou jsou generovány dvě fotonové mapy:

- Mapa pro difúzní povrchy s menší hustotu fotonů a vyššími úrovněmi energií fotonů. Jako hrubá aproximace intenzity světla ve scéně. Fotony jsou emitovány všemi směry a po dopadu zachytávány do mapy.
- Mapa pro odrazivé povrchy a kaustiky s vyšší hustotu fotonů a nižšími úrovněmi energií fotonů. Vytvořena emitováním fotonů směrem k odrazivým povrchům a zachytáváním dopadu odražených fotonů na difúzní povrch.

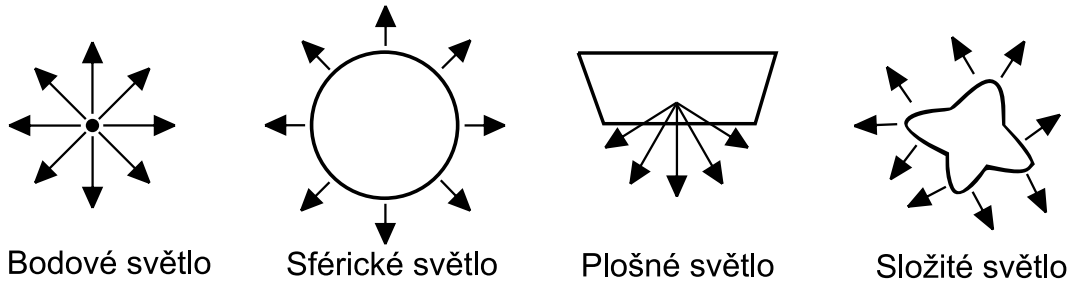


Obrázek 6: Vytváření fotonových map. Vlevo pro kaustiky, vpravo globální fotonová mapa

Možnost oddělení těchto map je velice výhodné, protože lze rozdělit jednotlivé části vykreslovací rovnice 1 a uložit je zvlášť do mapy. Tím se oddělí fotony s různou hustotou a intenzitami, což umožňuje oddělení jednotlivých částí vykreslovací rovnice a jejich řešení různými technikami. Fotonové mapy tak přináší variabilitu ve vykreslování scény.

2.4.1 První průchod – trasování fotonů

Pro každé světlo ve scéně vygenerujeme určité množství fotonů, mezi které rozdělíme celkovou sílu světla. Ta je dána tím, kolik energie dané světlo vyzařuje. Udává se několik typů světelných zdrojů, kde každý typ vyzařuje fotony jiným způsobem. Pro plošné světlo je vybrán náhodný bod na jeho povrchu a náhodný směr v hemisféře kolem tohoto bodu, který určuje směr šíření fotonu. U bodového světla jsou naopak fotony vysílány všemi směry a rozložení těchto směrů je uniformní.



Obrázek 7: Typy světelných zdrojů

Podle typu scény může být způsob šíření fotonů ze zdroje světla upraven. Ve scéně s malým počtem objektů se může stát, že většina fotonů vyslaných ze zdroje nedopadne na žádný povrch. Emitování těchto fotonů je tedy ztrátou času. Pro úpravu šíření se používá

projekčních map, které jsou reprezentovány bitmapami. Jde o mapu objektů z pohledu zdroje světla obsahující buňky, které určují, zda je v jejich směru objekt nebo ne. Ovlivňují tak, zda se mají fotony v daném směru emitovat. Se znalostí těchto map jsou emitovány fotony se zdroje světla. Po dopadu fotonu na povrch zjistíme z materiálu povrchu, jak velká část fotonu je pohlcena, odražena a jaká část se dostane pod povrch a bude se šířit tam.

Protože předpokládáme použití desítek tisíc až miliónů fotonů, které bude třeba často procházet během vykreslování, je nutné ukládat je do datové struktury stromu (nejčastěji vyvažovaný kd-tree), která nám zajistí rychlejší procházení a vyhledávání fotonů.

```
struct photon {
  float x, y, z;        // pozice fotonu
  char p[4];            // intenzity RGB slozek
  char phi, theta;      // smer dopadu fotonu
  short flag;           // priznak pro kd-strom
}
```

Výpis 2: Struktura pro uložení fotonu podle [4]

Hodnoty x, y, z udávají pozici fotonu ve scéně. Ta může být eventuálně komprimována do 24 bitů. Intenzity jednotlivých RGB složek jsou uloženy ve 4 bytech ve formě Wardova RGBE formátu, který ukládá RGB složky jako 3 byty a jeden byte slouží jako sdílený exponent. Ten umožňuje rozšíření rozsahu každé barevné složky a hlavně vyšší přesnost s nižšími paměťovými nároky. Dále jsou ve struktuře úhly dopadu fotonu na povrch ve sférických souřadnicích. Protože během vykreslování budou fotony procházeny mnohokrát a funkce jako cos() a sin() jsou poněkud náročné, je vhodné mít pro tyto úhly vyhledávací tabulku s výsledným směrem těmto úhlům odpovídajícím.

2.4.2 Druhý průchod – vykreslování

Pro vykreslování se používá klasické sledování paprsků, kdy vyšleme každým pixelem promítací roviny paprsek. Velikost osvětlení bodu, který protnul testovací paprsek, je dáno hustotou fotonů, které udávají vyzařování každého bodu. Kolem takového bodu v určitém poloměru získáváme fotony a výslednou hodnotu interpolujeme. Proto nepotřebujeme mít fotony v každém bodě povrchu, ale pár fotonů kolem testovaného bodu. Záleží také na volbě poloměru testované oblasti. Ten by měl být proporcionálně úměrný k počtu vygenerovaných fotonů. Malý poloměr vůči hustotě fotonů vede k vytváření zašumělého obrazu. Zvyšování počtu fotonů a prohledávání větší oblasti zlepšuje kvalitu výsledného obrazu, avšak navyšuje výpočetní čas.

Pro výpočet hustoty fotonů v konkrétním bodě povrchu se používá následující rovnice. Udává příspěvek odraženého záření, které je způsobené přímým zářením zdroje světla. Jedná se o nejdůležitější část odraženého záření, proto musí být vypočtena co nejpřesněji.

$$L_r(\mathrm{X},\omega) = \int_{\Omega} f_r(\mathrm{X},\omega_{\mathbf{i}},\omega) L_i(\mathrm{X},\omega_{\mathbf{i}}) \cos(\theta_i)\,\mathrm{d}\omega_{\mathbf{i}}\ , \tag{4}$$

kde Ω je hemisféra kolem bodu povrchu X, $\omega_{\mathbf{i}}$ je směr příchozího záření, ω je směr odchozího záření, f_r je obousměrná distribuční funkce odrazivosti (BRDF), L_i je velikost

příchozího záření, $\mathrm{d}\omega_\mathbf{i}$ je prostorový úhel dopadu příchozího záření hemisféry Ω a cos (θ_i) je faktor ovlivňující velikost příchozí záření. V této rovnici nahradíme příchozí záření L_i zářením E_i, které udává součet hodnot fotonů Φ_i na jednotkovou oblast A dané touto rovnicí:

$$E_i = \sum_{i=0}^{n} \frac{\Phi_i}{A} \ . \tag{5}$$

S prostorovým úhlem $\mathrm{d}\omega_\mathbf{i}$ můžeme vyjádřit diferenciální rovnici záření $\mathrm{d}E_i$ s využitím příchozího záření L_i jako:

$$\begin{aligned} \mathrm{d}E_i(\omega_\mathbf{i}) &= L_i(\omega_\mathbf{i})\cos(\theta_i)\,\mathrm{d}\omega_i \\ L_i(\omega_\mathbf{i}) &= \frac{\mathrm{d}E_i(\omega_\mathbf{i})}{\cos(\theta_i)\,\mathrm{d}\omega_i} \ . \end{aligned} \tag{6}$$

Po dosazení termínu příchozího záření v rovnici 4 a vykrácení termínů $\cos(\theta_i)\,\mathrm{d}\omega_i$ dostaneme výslednou rovnici pro výpočet odraženého záření v daném bodě X, které je dáno fotony v tomto bodě a vlastností povrchu danými BRDF funkcí f_r.

$$L_r(\mathrm{X}, \omega) = \int_\Omega f_r(\mathrm{X}, \omega_\mathbf{i}, \omega)\,\mathrm{d}E_i(\omega_\mathbf{i}) \ . \tag{7}$$

Pro výpočet odrazivých a lesklých ploch se nepoužívá výpočet z fotonových map pomocí integrálu, protože by bylo třeba velké množství fotonů. Pro ušetření paměti se používá trasování Monte Carlo metodou, která spočívá v testování náhodných vzorků a náhodném rozhodování, co se bude dít s paprskem po dopadu. Tento způsob výpočtu lesklých a odrazivých povrchů podává postačující výsledky i za použití nízkého množství testovacích paprsků.

Pro vykreslování kaustik je použita fotonová mapa pro kaustiky s vyšší hustotou fotonů. Z této mapy jsou jasně viditelná místa s velkou hustotou fotonů v určité oblasti (místa, kde se nacházejí kaustiky). Tyto data se použijí v rovnici 7 a výsledkem je velikost osvětlení pomocí kaustik v daném bodě. Přesnost výpočtu vyzařování povrchu udává počet fotonů, které vyšleme ze zdroje světla. Zvyšování počtu fotonů zlepšuje kvalitu obrazu, ale zpomaluje jeho výpočet, proto volba počtu fotonů záleží na aplikaci a na požadované kvalitě výsledku.

3 Ambient Occlusion

Okolní osvětlení (ambient illumination) je aproximace světla, které přichází do scény a odráží se od objektů ve scéně. Ambient occlusion (zastínění okolím) je útlum světla, ke kterému dochází, když je okolní osvětlení blokováno objektem nebo jeho částí v určitém bodě scény. Jedná se o hrubou simulaci globálního osvětlení ve scéně (dílčí efekt globálního osvětlení).

Jak okolní osvětlení, tak i zastínění jsou podstatné pro lepší pochopení tvarů a hloubky v reálném prostředí. Proto je i ve virtuálním prostředí snaha tento jev napodobit a přidat tak do scény ztmavení rohů objektů a kontaktní stíny. Principem zastínění je, že v každém bodě povrchu uvažujeme hemisféru (případně sféru) Ω, která jej obklopuje. Velikost zastínění v každém bodě povrchu je poté dáno plochou této hemisféry, do které nezasahuje žádný objekt scény, případně žádná část vlastní geometrie objektu.

Výslednou plochu zastínění hemisféry získáme tak, že z bodu P vedeme určitý počet paprsků, které slouží k testování, zda se v daném směru nenachází část modelu. V případě, že paprsek během svého putování neprotnul žádnou geometrii, je bod v tomto směru dostupný pro příchozí osvětlení. Výpočet velikosti zastínění v bodě P s normálou **n** je počítán jako:

$$A(\mathrm{P}, \mathbf{n}) = \frac{1}{\pi} \int_{\Omega} V(\mathrm{P}, \omega)\, \max(\mathbf{n} \cdot \omega, 0)\, \mathrm{d}\omega \,, \tag{8}$$

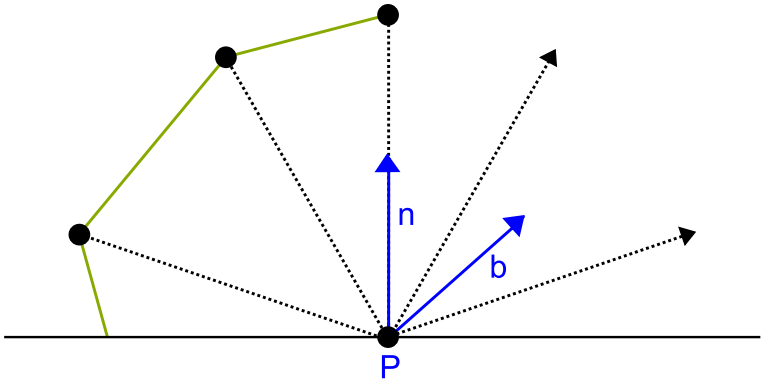
kde ω jsou všechny vektory v dané hemisféře, V je funkce viditelnosti, kde $V = 1$, když je v daném směru paprsku překážka a $V = 0$, když paprsek na překážku nenarazí. Skalární součin $\mathbf{n} \cdot \omega$ provádí váhování zastínění (kosinová distribuce) podle odklonu testovacího paprsku od normály a funkce max() zajišťuje testování pouze v hemisféře kolem normály. Protože integrujeme přes hemisféru, je třeba výsledek znormalizovat koeficientem $\frac{1}{\pi}$.

3.1 Výpočet zastínění objektů

Při popisu jednoho z prvních algoritmů výpočtu zastínění jsem vycházel ze zdroje [10], který je z roku 2004 a řeší výpočet zastínění v každém bodě povrchu scény. Hlavní myšlenkou realizace co nejpřesnějšího výpočtu zastínění je přípravný krok, kterým je třeba zastínění vypočítat a poté použít získané hodnoty během vykreslování. Máme-li objekt, u kterého chceme provést výpočet zastínění, musíme znát pro každý bod jeho povrchu dvě vlastnosti:

Dostupnost bodu P pro všudypřítomné osvětlení
: Udává velikost nezastíněné plochy hemisféry v bodě P, kterou přichází světlo k danému bodu. Jedná se o intenzitu osvětlení daného bodu v rozmezí 0 – 1 (0 znamená zastíněný bod, 1 znamená zcela osvětlený bod), která může být později zobrazena nebo použita společně s barevným obrazem scény.

Průměrný směr b nezastíněné části hemisféry
: Směr, kterým přichází nejvíce světla. Tento vektor **b** slouží pro získání barvy z environmentální mapy a ovlivňuje tak výslednou barvu bodu P.



Obrázek 8: Ukázka bodu P s normálou **n** a výsledným směrem nezastíněné hemisféry **b**

Na obrázku 8 znázorňují černé body vrcholy geometrie, které spojuje zelená čára představující trojúhelníky modelu. Tato geometrie obklopuje z poloviny bod P a způsobuje jeho zastínění. Vektor směru **b** a velikost dostupnosti bodu jsou hodnoty, které nejsou závislé na pozici světla. Jsou závislé pouze na geometrii, proto mohou být vypočteny v přípravné fázi a poté už jen používány ve fázi vykreslování. Tyto hodnoty je třeba vypočítat pro každý trojúhelník objektu. Poté se pro každý vrchol sítě uloží hodnota rovna průměru hodnot trojúhelníků, které jej obklopují. Při vykreslování je dáno zastínění v každém bodě trojúhelníku interpolací mezi hodnotami vrcholů.

Bod P získáme jako střed trojúhelníka a výpočet dostupnosti bodu a směru nezastíněné části probíhá následujícím způsobem. Pro každý bod P se vygenerují testovací směry paprsků (ukázka kódu 4) uniformně rozložené uvnitř jednotkové hemisféry, která je orientovaná kolem normály **n** bodu P. Ty sledujeme po jejich cestě z bodu P a zjišťujeme, zda narazily na jiný trojúhelník scény, což znamená blokování světla v tomto směru. Vektory paprsků, které neprotnuly žádnou geometrii sčítáme a ve výsledku vektor znormalizujeme. Tak získáme jednotkový vektor nezastíněného směru **b**.

```
pro kazdy trojuhelnik sceny
{
  vypocitej stred trojuhelniku
  vytvor mnozinu paprsku ve smeru hemisfery
  Vector nezastineny_smer = Vector(0, 0, 0);
  int pocet_nezastinenych = 0;
  pro kazdy paprsek
  {
    kdyz (paprsek neprotnul zadny trojuhelnik)
    {
      nezastineny_smer += paprsek.smer; // pricti aktualni smer paprsku
      ++pocet_nezastinenych;            // inkrementuj pocet prispevku
    }
  }
  nezastineny_smer = normalize(nezastineny_smer);
  dostupnost_bodu = pocet_nezastinenych / pocet_paprsku;
}
```

Výpis 3: Pseudokód výpočtu dostupnosti a vektoru směru maximálního osvětlení bodu podle [10]).

Vytvoření množiny paprsků v hemisféře spočívá v náhodném generování vektoru se souřadnicemi x, y, z. Ten se generuje do té doby, dokud neleží uvnitř jednotkové hemisféry orientované kolem normály **n**. Je potřeba vygenerovat náhodné hodnoty ve všech třech souřadných osách uvnitř jednotkové sféry (od -1 do 1). U vygenerovaného vektoru se otestuje jeho velikost a orientace vůči normále trojúhelníku. Pokud projde těmito testy, navrátí se znormalizovaný vektor.

```
while (true) {
  //nastav nahodne hodnoty souradnic smeru paprsku v rozmezi −1 az 1
  x = RandomFloat(−1, 1);
  y = RandomFloat(−1, 1);
  z = RandomFloat(−1, 1);

  if (x ∗ x + y ∗ y + z ∗ z > 1) continue; //pokud je mimo jednotkovou sferu vygeneruj novy

  if (dot(Vector(x, y, z), n) < 0) continue; //pokud je paprsek v opacne hemisfere vygeneruj
      novy

  return normalize(Vector(x, y, z)); // vrat normalizovany vektor
}
```

Výpis 4: Metoda generující uniformně rozložené paprsky v orientované hemisféře podle [10]).

3.2 Vykreslení pomocí vypočteného zastínění

Pro vykreslení zastínění je třeba poslat na grafickou kartu do vertex shaderu spolu s informacemi o vykreslovaném trojúhelníku i hodnoty dostupnosti a nejméně zastíněného směru. Tyto hodnoty zastínění jsou, jak již bylo zmíněno, při odeslání do fragment shaderu interpolovány mezi jednotlivými vrcholy trojúhelníku. Ty poté využijeme k určení výsledné barvy pixelu obrazu.

Takový výpočet zastínění se řadí do fyzikálně přesnějších. Výpočet scény se 150000 trojúhelníky, kde na každý bylo použito 512 testovacích paprsků, trval v roce 2004 na procesoru AMD Athlon 1800XP přibližně 4 minuty.

V další kapitole se budu věnovat přístupům k výpočtu zastínění, které jej řeší v reálném čase. Protože je toto téma v posledních pár letech populární a snad každé studio má snahu zavést tento prvek do své hry nebo aplikace, musí být výpočty co nejrychlejší, aby nezpomalovaly vykreslovací smyčku. Na druhou stranu by neměl příliš zasahovat do scény, ale jen dotvářet dojem realističnosti. Proto se bude jednat o pouhé aproximace zastínění s docela překvapivými výsledky.

4 Zastínění v reálném čase

4.1 Screen-Space Ambient Occlusion

Tato metoda je nejjednodušší a pravděpodobně nejrozšířenější metodu výpočtu zastínění okolím s přijatelnými výsledky. Vyvinul ji vývojář Cryteku, Vladimir Kajalin, v roce 2007, kdy byla poprvé použita ve hře Crysis. K realizaci tohoto algoritmu se využívá pouze depth bufferu z pohledu pozorovatele. Řešení se rychle uchytilo a bylo použito ve hrách jako *S.T.A.L.K.E.R.: Clear Sky(2008)*, *ARMA 2(2009)*, *The Elder Scrolls V: Skyrim(2011)*.

Myšlenka algoritmu je taková, že se využívá depth bufferu jako aproximace geometrie viditelného obrazu. Z těchto hodnot lze vypočítat zastínění ve screen-space (prostor pixelů obrazu). Zajišťuje to možnost přesunu většiny operací z procesoru a provádění výpočtu zastínění na grafické kartě, který je navíc nezávislý na složitosti scény.

Přístup byl zpočátku založen na testování okolních pixelů depth bufferu, který dokázal vypočítat ztmavení v oblastech okrajů objektů a získat tak jejich siluety. Pro docílení efektu zastínění okolím byl výpočet omezen na blízké objekty a po několika iteracích a optimalizacích byl vytvořen algoritmus nazvaný Screen-Space Ambient Occlusion.

4.1.1 Algoritmus

Princip algoritmu je takový, že pro každý pixel obrazu je v jeho okolí testováno několik bodů v prostoru kamery. Ty jsou projektovány do screen-space a jejich hloubka je porovnána s hloubkou uloženou v depth bufferu. Tím se zjistí, zda je testovaný bod nad povrchem (hloubka menší než hodnota v bufferu) nebo pod ním (hloubka větší než hodnota v bufferu).

Pro snížení počtu testovacích vzorků je použita randomizace vzorků. Testovací vzorky jsou náhodně rozděleny kolem středu testovací sféry bodu P. Toho je dosaženo v shader programu na grafické kartě pomocí funkce reflect, která vrací odražený paprsek daný normálou **n** a incidentním paprskem **i**. Jako incidentní paprsek **i** je použit testovací vektor z vygenerované sféry. Normálu **n** pro odražení tohoto paprsku představuje náhodně vygenerovaný vektor. Tímto řešením se zamezí vytváření fragmentů jako jsou pásy jedné barvy tzv. banding.

```
vec3 reflect(vec3 i, vec3 n) {
  return i - 2 * dot(i, n) * n;
}
```

Výpis 5: Funkce pro výpočet odraženého paprsku

Protože algoritmus netestuje body okolí v hemisféře, ale v celé sféře kolem bodu P, padne velké množství testovacích bodů na rovných plochách pod povrch. To způsobí zatemnění rovných ploch, kde by nemělo k žádnému zastínění docházet. Výsledkem je zatemnění celého obrazu v místech objektů scény a obraz je zašedlý.

Obrázek 9: Scéna zastínění okolím vypočtená SSAO metodou společnosti Crytek. Převzato z [5]

Pro odstranění šedých ploch je třeba výpočet zastínění upravit. Pokud velká část vzorků spadá pod povrch plochy, není nutné tyto vzorky generovat a počítat s nimi. Proto lze tyto testovací vzorky vypustit a počítat zastínění bodu P pouze v hemisféře orientované kolem jeho normály **n**.

4.1.2 Normálově orientovaná hemisféra

Abychom mohli generovat testovací body, které budou uvnitř hemisféry orientované kolem normály **n** bodu P, je třeba znát tyto normály. Ty lze získat dvěma způsoby:

V přípravném kroku spolu s generováním depth bufferu
Scéna se nejprve vykreslí z pohledu pozorovatele do bufferu s RGBA texturou, kde RGB složky tvoří normály objektů transformované normálovou maticí. Složka A reprezentuje linearizovanou hloubku pixelu.

```
VS
void main() {
  vec4 MVP_pos = MVP * vec4(inPosition, 1.0);
  vec4 MV_pos = MV * vec4(inPosition, 1.0);
  normal = N * inNormal; //transformace normaly normalovou matici

  depth = MV_pos.z;
  depth = (-depth - near) / (far  - near);  // linearizace  hloubky

  gl_Position  = MVP_pos;
}

FS
void main() {
  color  = vec4(normal.x, normal.y, normal.z, depth);  //vystup do textury  (RGB -
       normala, A - hloubka)
}
```

Výpis 6: Shader programy pro získání normály a hloubky pixelu

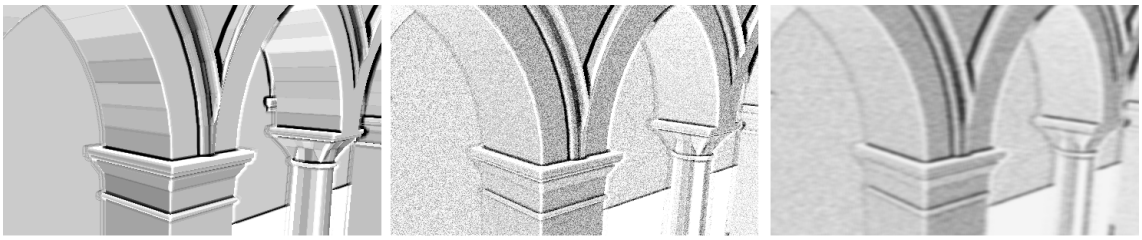
Toto řešení je použito v implementaci SSAO praktické části.

Během výpočtu zastínění

Při výpočtu zastínění lze vypočítat normálu v pixelu pomocí derivací dx, dy ve směrech x a y nad depth bufferem.

Toto řešení je použito v implementaci HBAO praktické části.

Dále musí být vygenerovány testovací body v hemisféře (stejně jako v kapitole 3.1). Každý tento bod je při výpočtu zastínění opět odražen kolem vygenerované normály funkcí reflect pro zamezení vytváření artefaktů, které vyvolávají umělý dojem. Navíc musí být u odraženého paprsku proveden test na jeho orientaci vůči normále, zda leží ve správné hemisféře. Odražené paprsky maskují artefakty, ale přidávají do obrazu šum, který je nutné nakonec minimalizovat.



Obrázek 10: (a) Obraz s jednolitými plochami. (b) Obraz s náhodně odraženými vzorky odstraňující artefakty. (c) Druhý obraz s použitím rozmazání

Zašumělý obraz zastínění se vykreslí pomocí fragment shaderu do textury. Ta je poslána do shaderů gaussovského rozostření, kde je každý pixel rozostřen maskou velikosti 5x5. Rozostření je provedeno v horizontálním a vertikálním směru zvlášť (viz 4.3.4). Výsledek rozostření je nakonec spojen s barevným obrazem scény. Posloupnost akcí pro výpočet zastínění a vykreslení obrazu je následující:

1. Vykresli scénu a ulož linearizovanou hloubku a normály pixelů do textury.
2. Vykresli polygon přes celou obrazovku SSAO shaderem a výsledek ulož do textury.
3. Proveď rozostření textury v horizontálním směru.
4. Proveď rozostření textury ve vertikálním směru.
5. Smíchej rozostřený výsledek zastínění s barevným obrazem scény nebo vykresli pouze obraz zastínění.

V roce 2008 společnost NVIDIA přišla s metodou, která je založena na hledání linie horizontu kolem testovaného bodu P jako algoritmus HBAO[4.3]. Tato metoda pracuje pouze ve screen-space (2D), kde se provádí sledování paprsku testovacího směru. Odpadá tak potřeba sledovat paprsky ve view-space (3D) a poté je projektovat do screen-space.

4.1.3 Výsledky aplikace

Výsledné časy vykreslovací smyčky s SSAO algoritmem. Jedná se o průměrný čas každé úlohy na jeden snímek aplikace. Časy jsou měřené na mobilní grafické kartě NVIDIA GeForce GT 740M.

Počet směrů	Vygenerování depth a normal bufferu	Výpočet zastínění	Výpočet barevného obrazu	Celkem	fps
8	2,05 ms	0,95 ms	3,12 ms	6,12 ms	74
32	2,08 ms	2,68 ms	3,19 ms	7,95 ms	61
48	2,01 ms	3,75 ms	3,18 ms	8,94 ms	49

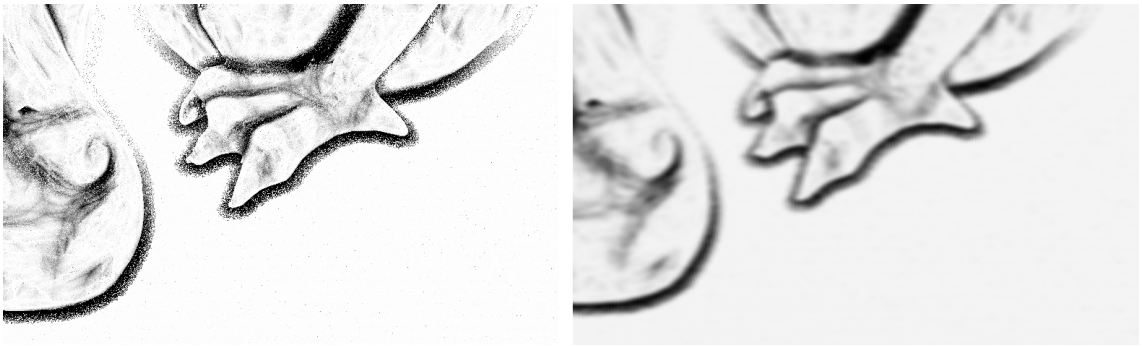
Tabulka 1: Průměrné časy jednotlivých částí vykreslovací smyčky pro jeden snímek aplikace. SSAO algoritmus v rozlišení 800px $\times$ 600px.

Počet směrů	Vygenerování depth a normal bufferu	Výpočet zastínění	Výpočet barevného obrazu	Celkem	fps
8	3,13 ms	2,15 ms	4,09 ms	9,37 ms	65
32	3,08 ms	6,23 ms	4,43 ms	13,74 ms	48
48	3,15 ms	8,75 ms	4,02 ms	15,92 ms	38

Tabulka 2: Průměrné časy jednotlivých částí vykreslovací smyčky pro jeden snímek aplikace. SSAO algoritmus v rozlišení 1366px $\times$ 768px.

Počet směrů	Vygenerování depth a normal bufferu	Výpočet zastínění	Výpočet barevného obrazu	Celkem	fps
8	4,36 ms	5,06 ms	5,24 ms	14,66 ms	40
32	4,18 ms	15,37 ms	5,34 ms	24,89 ms	26
48	4,16 ms	21,62 ms	5,28 ms	31,06 ms	23

Tabulka 3: Průměrné časy jednotlivých částí vykreslovací smyčky pro jeden snímek aplikace. SSAO algoritmus v rozlišení 1920px $\times$ 1080px.

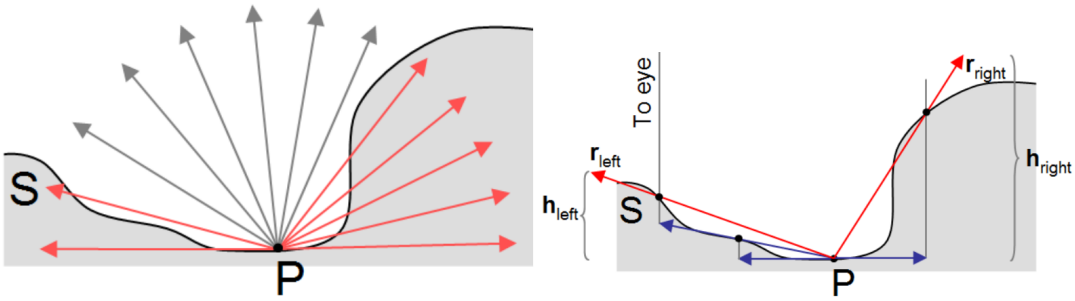


Obrázek 11: Vykreslení části modelu Stanford dragon metodou SSAO s 8 testovacími paprsky. Vlevo bez rozostření, vpravo s gaussovským rozostřením

Obrázek 12: Vykreslení části modelu Stanford dragon metodou SSAO s 48 testovacími paprsky. Vlevo bez rozostření, vpravo s gaussovským rozostřením

4.2 Horizon-Split Ambient Occlusion

HSAO je algoritmus z roku 2007 představený společností NVIDIA (zdroj [6]), který vychází z hledání horizontů kolem testovaného bodu P. Nalezením horizontu v určitém směru rozděluje hemisféru na dvě části. Jedna je odkrytá a vhodná pro testování zastínění, druhá část je pod úrovní horizontu a její testovací paprsky lze proto přeskočit. Navíc lze zastínění pod horizontem vypočítat zjednodušeně v 1D prostoru (místo 2D).



Obrázek 13: (a) Červené paprsky protínají povrch S v okolí bodu P. (b) Výpočet velikosti horizontu postupným přikláněním paprsků směrem k pozorovateli. Převzato z [6]

Velikost horizontu v určitém směru H(θ) je výška od bodu P po konec paprsku horizontu, který začíná v bodě P a protíná horizont. Hodnota výšky je normalizovaná na rozsah od 0 do 1. Směr těchto paprsků je zpočátku nastaven na tečnu roviny bodu P (kolmá k normále **n**) a postupně se přiklánějí směrem k pozorovateli, dokud je délka paprsku menší než radius testování r. Rovnice pro výpočet tohoto zastínění je následující:

$$A = 1 - \frac{1}{\pi}\left(\int_{\theta=0}^{2\pi} T(\theta)\,\mathrm{d}\theta + \int_{\theta=0}^{2\pi}\int_{z=H(\theta)}^{1} V_\omega(\mathbf{n}\cdot\omega)\,\mathrm{d}\omega\right) . \tag{9}$$

Výpočet se skládá ze dvou částí. První je 1-dimenzionální integrál, který se nazývá zastínění horizontem a počítá jej ve směru θ. Autoři vychází z vlastnosti, že $T(\theta) = H(\theta)^2$, a proto pro každý směr θ není třeba testování paprsků v hemisféře. To umožňuje soustředit se na testování paprsků v druhé části a přitom znát zastínění horizontem. Druhá část je integrál ve 2D, kde se provádí výpočet funkce viditelnosti pro nezastíněnou část hemisféry danou horizonty. Tato část rovnice se nazývá zastínění normály. Rovnice zastínění horizontem slouží pro spojité povrchy a zastínění normály pro výpočet zastínění mezi objekty (kontaktní stíny).

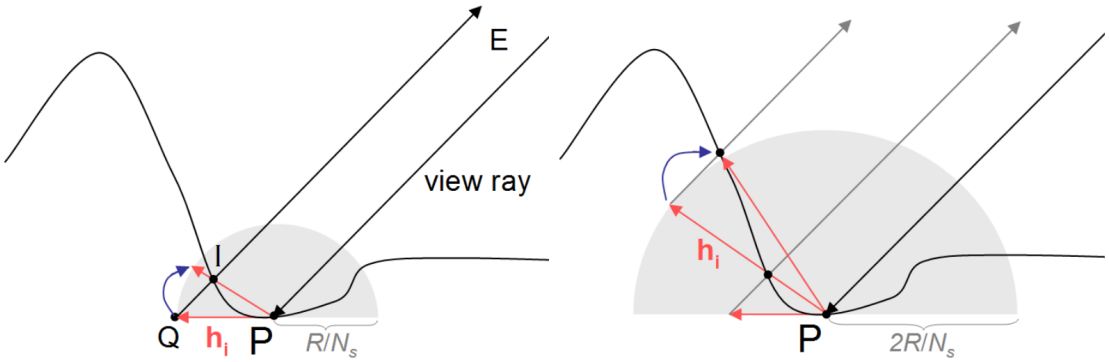
Algoritmus využívá pro výpočet depth bufferu a normal bufferu scény, kde se znalostí normály každého pixelu obrazu a hloubek okolních pixelů je možné vypočítat výšky horizontů a funkce viditelnosti. Přestože je algoritmus interně založený na výpočtu zastínění ve view-space (prostor kamery), probíhá hledání horizontu na přímce projektované do screen-space (obraz po aplikaci projekce). Jedná se o optimalizaci sledování paprsku (ray-marching) podél přímky horizontu z testovací hemisféry.

Při procházení pixelů obrazu je nejprve nutné vypočítat pozici pixelu v prostoru kamery. Dále se vygenerují paprsky orientované kolem normály **n** bodu P v náhodných směrech θ. Pro každý takový paprsek se v jeho směru θ prochází depth buffer a hledá se horizont $H(\theta)$. Následně je vypočtena hodnota zastínění horizontem $T(\theta)$ a paprsky normály, směřující nad horizonty, udávají zbývající část rovnice zastínění. Takto je zpracován každý bod viditelného obrazu scény.

4.2.1 Hledání horizontu a výpočet jeho příspěvku k zastínění

Pro nalezení horizontu ve směru θ je potřeba nalézt směr vektoru, který vychází z bodu P a je tečnou k horizontu, tedy bodu maximálně omezujícím hemisféru v tomto směru. Nejprve zvolíme paprsek horizontu jako tečnu roviny **t** v bodě P určené její normálou **n** (h = (P, **t**)). Jeho délka je omezena velikostí testovací hemisféry. Koncový bod Q paprsku nyní projektujeme do screen-space prostoru pozorovatele a porovnáme jeho hloubku s příslušnou v depth bufferu. Pokud je hodnota depth bufferu blíže pozorovateli (horizont stoupá) a jsou splněny další podmínky, je paprsek horizontu nastaven tímto směrem (přikloněn k pozorovateli) a jeho velikost je nastavena na velikost poloměru hemisféry r. Podmínkami pro přijetí nového horizontu jsou:

- koncový bod paprsku horizontu Q je ve vzdálenosti maximálně r
- paprsek horizontu má správný směr (směřuje stejným směrem jako tečna roviny bodu P)



Obrázek 14: Hledání horizontu v HSAO algoritmu. Převzato z [6]

Pro jeden směr θ je tento proces opakován v N_s krocích.

```
H(θ) :=
  h ← T //nastav horizont na tecnu roviny v bode P
  for i ← 1..N_s //hledej horizont v krocich N_s
    do Q ← P + i R/N_s h //nastav koncovy bod paprsku horizontu
      if ISBELOW(Q, S) //Q je pod povrchem S //pokud je pod povrchem S
        then I ← INTERSECT(Q − E, S) //nastav I jako projekci Q na povrch S
          if LENGTH(I − P) < R and DOT(I − P, T) > 0 //pokud je delka paprsku mensi nez
              polomer a ma spravny smer
            then h ← NORMALIZE(PROJ_{n,t} (I − P)) //nastav horizont
  return DOT(h, n) //vrat velikost horizontu
```

Výpis 7: Pseudokód výpočtu velikosti horizontu podle [6]

Tento algoritmus slouží pro výpočet horizontu v jednom směru od bodu P. Abychom získali hodnoty horizontů v celé hemisféře a z nich velikost zastínění bodu P horizontem, je třeba projít všechny směry kolem normály **n**. Počet těchto směrů je N_d a celkový příspěvek zastínění horizontem udává následující rovnice:

$$\begin{aligned} O_T &= \frac{1}{\pi}\int_{\theta=0}^{2\pi} T(\theta)\,\mathrm{d}\theta \\ &= \frac{1}{\pi}\int_{\theta=0}^{2\pi}\int_{z=0}^{H(\theta)} (\mathbf{n}\cdot\omega)\,\mathrm{d}\omega\ . \end{aligned} \tag{10}$$

Pro výpočet integrálu je použito cylindrických souřadnic (r,θ,z), kde $\mathrm{d}\mathbf{w} = \mathrm{d}z\,\mathrm{d}\mathbf{w}r$ a r = 1. Část hemisféry pod úrovní horizontu je nahrazena hemisférami s rozměrem $H(\theta_i)$, které zaujímají úhel $\alpha = \theta_{i+1} - \theta_i = 2\pi/N_d$ kolem normály. Výpočet hodnoty zastínění O_T lze vyjádřit takto:

$$\begin{aligned} O_T &= \frac{1}{\pi}\sum_{i=1}^{N_d}\int_0^{H(\theta_i)}\int_{\theta_i}^{\theta_{i+1}} (\mathbf{n}\cdot\omega)\,\mathrm{d}\theta\,\mathrm{d}z \\ &= \frac{\alpha}{\pi}\sum_{i=1}^{N_d}\int_0^{H(\theta_i)} (\mathbf{n}\cdot\omega)\,\mathrm{d}z \\ &= \frac{2}{N_d}\sum_{i=1}^{N_d}\int_0^{H(\theta_i)} z\,\mathrm{d}z \\ &= \frac{1}{N_d}\sum_{i=1}^{N_d} H^2(\theta_i)\,\mathrm{d}z\ . \end{aligned} \tag{11}$$

4.2.2 Zastínění normály

Pro každý paprsek horizontu $\mathbf{h_i}$ je ve zbylé části hemisféry bodu P vygenerováno N_n paprsků, testujících kolizi s objekty v těchto směrech. Každý z těchto paprsků končí v určité výšce z_k nad bodem P a tvoří výseč v hemisféře značenou Π_k. Zastínění normály ve směru tečny $\mathbf{t_i}$ je vypočteno jako:

$$\begin{aligned} O_{N,i}(z_k) &= \frac{1}{\pi}\int_{\theta=\theta_i}^{\theta_{i+1}}\int_{z_k}^{z_{k+1}} V(\omega)(\mathbf{n}\cdot\omega)\,\mathrm{d}\omega \\ &= \frac{\alpha}{\pi}\int_{z_k}^{z_{k+1}} V(\omega)(\mathbf{n}\cdot\omega)\,\mathrm{d}\omega \\ &= \frac{2}{N_d}\int_{z_k}^{z_{k+1}} V(\omega)z\,\mathrm{d}\omega \\ &= \frac{1}{N_d}(z_{k+1}^2 - z_k^2)V(\omega)\ . \end{aligned} \tag{12}$$

Pro každou výseč Π_k se provede výpočet funkce viditelnosti $V\omega)$ ve směru paprsku r_k. Ten je rozdělen na N_s kroků, ve kterých se postupně získávají hodnoty z depth bufferu. Z těchto hodnot se zjišťuje, zda je v daném směru překážka a bod bude zastíněn. Výsledky funkce viditelnosti z každého kroku N_s a v každém směru N_d se sečtou a vydělí celkovým počtem. To udává celkové zastínění normály dané rovnicí:

$$O_N = \sum_{i=1}^{N_d} \sum_{k=1}^{N_n} \frac{V(\omega)}{N_n N_d} \ . \tag{13}$$

Jakmile jsou vypočteny příspěvky zastínění horizontem O_T a zastínění normály O_N, tak se provede výpočet celkové hodnoty zastínění bodu P jako:

$$A = 1 - (O_T + O_N) \ . \tag{14}$$

4.2.3 Parametry zastínění

Vlastnosti, které ovlivňují rychlost výpočtu a kvalitu jeho výsledku jsou poloměr výpočtu zastínění, počet testovacích směrů kolem normály **n**, počet testovacích směrů nad horizontem a počet kroků při sledování testovacího paprsku. Podrobnější popis parametrů:

Poloměr vlivu zastínění r
: Tento parametr určuje velkost poloměru testovací hemisféry. Zvětšování poloměru vede k situaci, kdy je třeba pro větší plochu více paprsků pro zachování kvality. Také to způsobuje větší míru zastínění scény, protože bude do testovací hemisféry zasahovat více objektů.

Počet směrů N_d
: Parametr N_d udává, kolik směrů zastínění se bude počítat kolem normály $\mathbf{n}$ a bude tvořit horizont $H(\theta)$ na hemisféře.

Počet normálových paprsků N_n
: Tato hodnota určuje, kolik paprsků bude vygenerováno ve zbývající části hemisféry nad horizontem, které slouží pro testování zastínění jiným objektem scény.

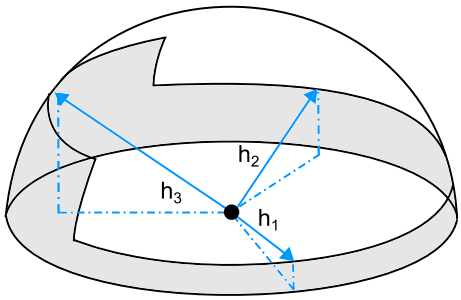
Počet kroků N_s
: Poslední parametr je počet kroků, který se používá u hledání horizontu a testování paprsků s objekty. Určuje počet kroků, které musí udělat každý vyslaný paprsek u obou metod. Hlavně tímto parametrem je ovlivňován výsledek algoritmu a jeho rychlost.

Obrázek 15: (a) Zastínění horizontem (b) Zastínění normály (c) Výsledné zastínění získané algoritmem HSAO složené z hodnot zastínění (a) a (b). Převzato z [6]

4.3 Horizon-Based Ambient Occlusion

4.3.1 Algoritmus

Algoritmus HBAO (podle [7]) je založen, stejně jako SSAO na znalosti depth a normal bufferu scény. Velikost zastínění u této metody je opět dána poměrem zastíněné plochy hemisféry v aktuálním bodě k její celkové ploše. Tato hodnota se aproximuje procházením depth bufferu v několika směrech od bodu P a hledáním nejnižších hodnot z-bufferu, tedy hodnot nejbližších pozorovateli. Tyto hodnoty jsou nazývány horizonty a představují výseč na testovací hemisféře.



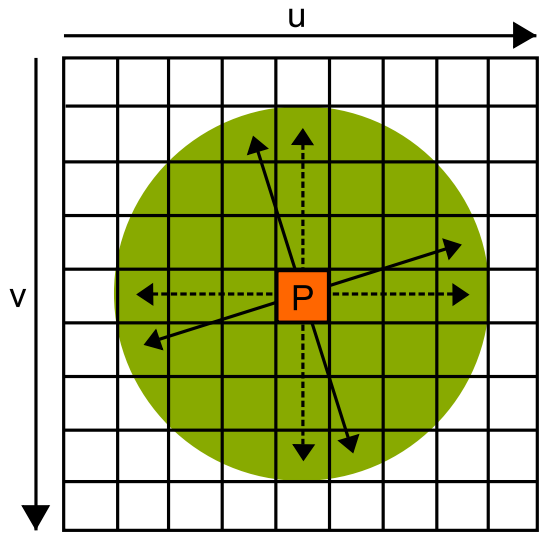
Obrázek 16: Vykrojení hemisféry horizonty tří testovaných směrů

Přestože se jedná o screen-space metodu, je výpočet zastínění prováděn ve view-space tedy v prostoru kamery. Pro každý procházený bod obrazu (aktuální i testovaný) je tedy nutné provést jeho rekonstrukci do view-space. Známe-li nastavení projektivní matice a uv souřadnice aktuálního pixelu obrazovky, můžeme provést transformaci ze screen-space do view-space jako:

```
vec3 GetViewPos(vec2 uv) {
  // ziskani hloubky z textury a prevod rozsahu (z 0 -> 1 na -1 -> 1)
  float depth = texture(depth_map, uv).r * 2.0 - 1.0;

  // prevod uv souradnic (z 0 -> 1 na -1 -> 1)
  uv = uv * 2.0f - 1.0f;
  // ziskani screen-space pozice
  //  a pridani hloubky jako z-ove souradnice
  vec4 pos_SS = vec4(uv, z, 1.0f);
  // vynasobeni screen-space pozice inverzni projektivni matici
  // - prevod do view-space
  vec4 pos_VS = inverse(projection_matrix) * pos_SS;

  // transformace z homogennich souradnic
  return pos_VS.xyz / pos_VS.w;
}
```

Výpis 8: Rekonstrukce view-space pozice

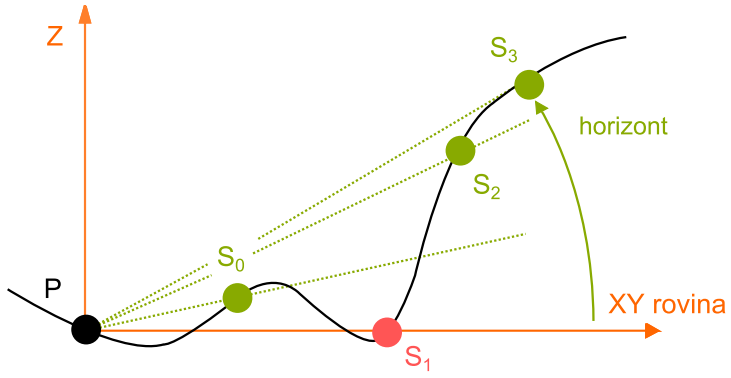
Pro každý bod P, kde počítáme zastínění opět procházíme okolní body v určitých směrech. Počet směrů je předem daný a jsou uniformně rozloženy kolem referenčního bodu P. Každý směr je rozdělen na několik částí, po kterých procházíme depth buffer. Směry testovacích paprsků se pro každý pixel otočí kolem P o náhodný úhel a navíc se posunou o náhodnou hodnotu po uv souřadnicích, což zabraňuje již známým artefaktům a vytváří v obraze šum.



Obrázek 17: Ukázka 4 testovacích paprsků otočených o náhodný úhel kolem P

Každý takto vygenerovaný paprsek prochází depth buffer scény v daném směru od bodu P. Máme-li definovaný poloměr testovací oblasti r, pak pro x vzorků vypočteme krok s, se kterým se bude procházet depth buffer ve směru paprsku jako $s = r/x$.

Při hledání horizontu v aktuálním směru se používá metody ray marching. Na začátku si uložíme hodnotu horizontu jako maximální možnou hloubku. Poté procházíme zvolený směr po krocích s a hledáme minimální hodnotu hloubky, tedy maximální horizont a vypočteme vektor **h**, který k němu směřuje z bodu P.



Obrázek 18: Ukázka ray marchingu při hledání horizontu

Pro výpočet zastínění potřebujeme znát vektor tečny roviny **t** v bodě P, který spolu s úhlem k horizontu bude udávat příspěvek zastínění v daném směru kolem normály. Tečnu můžeme získat pomocí převodu sousedních hodnot bodu P z depth bufferu do prostoru kamery, ze kterých získáme derivace ve směrech u a v. Těmito hodnotami se vynásobí směr kroku pří hledání horizontu, čímž získáme vektor tečny v daném bodě:

```
// potrebujeme znat rozliseni vykreslovaneho obrazu
vec2 res = vec2(width, height);

// prevracena hodnota rozliseni nam udava velikost jednoho pixelu
//    pri prochazeni uv float souradnic textury od 0 do 1, ktera
//    je potrebna pro ziskani sousednich hodnot bodu P
vec2 inv_res = 1.0f / res;

// ziskame okolni hodnoty bodu ve view-space kolem bodu P s inUV
//    souradnicemi na obrazovce
vec3 p_l = GetViewPos(inUV + vec2(-inv_res.x, 0));
vec3 p_r = GetViewPos(inUV + vec2(inv_res.x, 0));
vec3 p_t = GetViewPos(inUV + vec2(0, inv_res.y));
vec3 p_b = GetViewPos(inUV + vec2(0,-inv_res.y));

// vypocteme minimalni diference ve smerech u a v od bodu P
//    jako minimalni vzdalenost okolnich bodu od bodu P
vec3 dPdu = MinDiff(P, p_l, p_r);
vec3 dPdv = MinDiff(P, pB, pT) * (res.y * inv_res.x);

// tecnu ziskame pomoci vypoctenych derivaci ve smerech uv
//    a smeru kroku delta_uv
vec3 t = delta_uv.x * dPdu + delta_uv.y * dPdv;
```

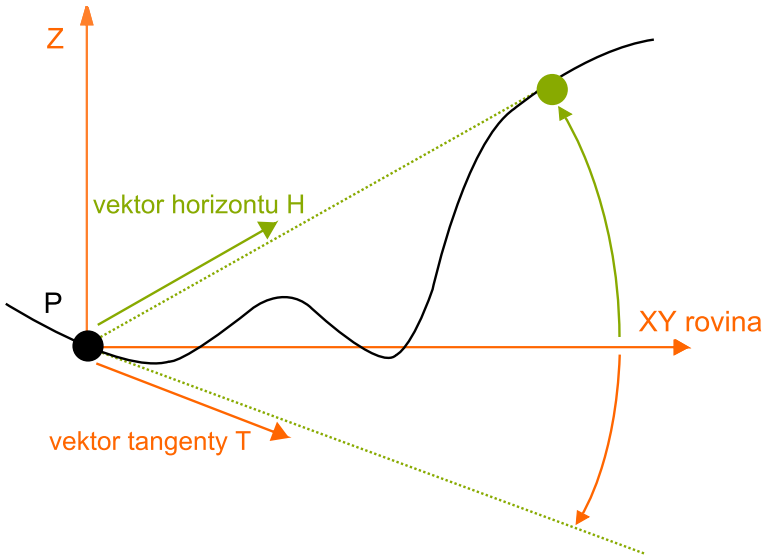
Výpis 9: Výpočet tečny bodu P z hodnot jeho okolí

Nyní známe pro bod P jeho tečnu **t** a vektor horizontu **h**. Potřebujeme vypočítat velikosti úhlů α_t a α_h, které tyto vektory svírají s rovinou xy, která je kolmá k pozorovateli. Využijeme goniometrické funkce arctan() na vektory takto:

$$\begin{aligned} \alpha_t &= \arctan\left(\frac{\mathbf{t}.z}{\|\mathbf{t}.xy\|}\right) \\ \alpha_h &= \arctan\left(\frac{\mathbf{h}.z}{\|\mathbf{h}.xy\|}\right) . \end{aligned} \tag{15}$$

Hodnota zastínění pro jeden směr kolem normály bodu P je dán rozdílem sinů úhlu horizontu a úhlu tečny bodu P. Pro výsledné zastínění v bodě P sečteme příspěvky zastínění ze všech směrů kolem normály a hodnotu vydělíme počtem směrů. Při procházení směru a hledání horizontu v každém kroku testujeme hodnotu depth bufferu, zda není aktuálním horizontem. Pokud je hodnota aktuálním maximem, přičteme k zastínění daného směru již zmiňovaný rozdíl sinů úhlů horizontu a tečny:

$$A = \sin(\alpha_h) - \sin(\alpha_t) . \tag{16}$$

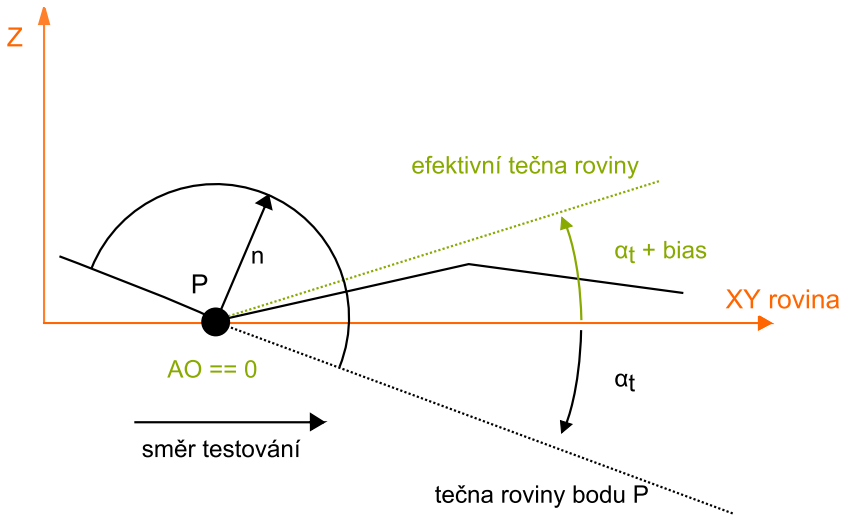


Obrázek 19: Vektor horizontu a tečna udávající hodnotu zastínění

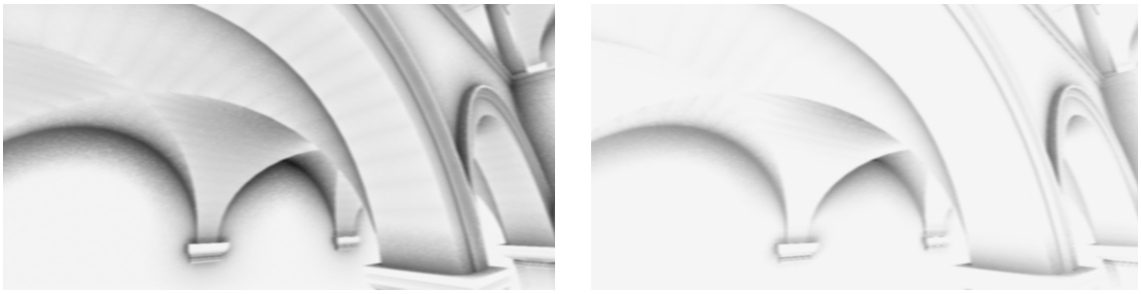
4.3.2 Problém nízké teselace

Tento problém nastává v případě, když provádíme výpočet zastínění v místě větší změny geometrie (např. mezi dvěma trojúhelníky). V tomto místě se mění normála a tedy i tečná rovina. Tečna roviny směřuje dovnitř geometrie a vzniká tak chybné zastínění. Řešení problému spočívá ve vytvoření efektivní tečny jejím posunem směrem k normále, tedy

přičtením úhlu (tzv. biasu) k aktuální tečně roviny. Tento posun roviny zamezí vzniku nesprávného zastínění.



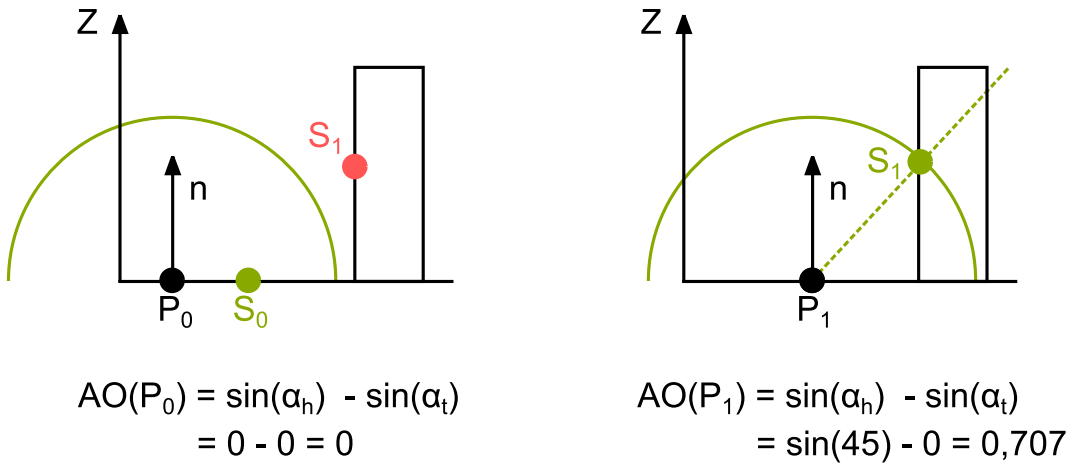
Obrázek 20: Efektivní tečna roviny bodu P po přičtení biasu



Obrázek 21: Vliv biasu na zastínění. Vlevo obraz bez biasu, vpravo nastaven bias na 20 stupňů

4.3.3 Problém skokové změny zastínění

Omezení výpočtu zastínění jen na oblast hemisféry kolem bodu P vede k problému neznalosti scény. Mezi dvěma sousedními body v rovině, kde provádíme výpočet zastínění, může dojít ke skokové změně této hodnoty. Bod P_0 ve své hemisféře nenarazí na žádnou překážku, ale hemisféra sousedního bodu P_1 již překážku objeví a vzniká tím vysoká změna zastínění mezi dvěma pixely výsledného obrazu.



Obrázek 22: Odhalení překážky bodem P_1

Pro zamaskování tohoto jevu je třeba nahradit tyto skoky funkcí útlumu, napodobující zatemnění (tzv. obscurance podle [11]), které zjemní přechod mezi velkými změnami intenzity osvětlení:

$$W(r) = 1 - r^2, r = \frac{\|\mathrm{S} - \mathrm{P}\|}{r_p}\ , \tag{17}$$

kde W je funkce útlumu pro testovaný bod S (potenciální nový horizont) a r je jeho vzdálenost od bodu P upravená velikostí testovací hemisféry $\mathrm{r_p}$ v bodě P. Tento útlum se neaplikuje na výsledné zastínění, ale násobíme jím každý testovaný bod, který má úhel horizontu větší než maximální a stává se tak aktuálním horizontem. Metoda se nazývá per-sample attenuation (útlum pro každý vzorek).

```
// prochazeni smeru v num_samples krocich velikosti delta_uv
for (int i = 1; i <= num_samples; ++i) {
  // pricteni kroku delta_uv k uv souradnicim
  uv += delta_uv;

  // ziskani bodu S k testovani
  s = GetViewPos(uv);

  // tangens uhlu vektoru horizontu
  tan_s = Tangent(p, s);

  // vypocet vzdalenosti od bodu P
  d2 = Length(s - p);

  // test jestli je S uvnitr testovaci hemisfery
  //   a jestli bude novym horizontem
  if (d2 < r2 && tan_s > tan_h) {
    float sin_s = TanToSin(tan_s);

    // vynasobeni zastineni utlumem zavislym na vzdalenosti S od P
```

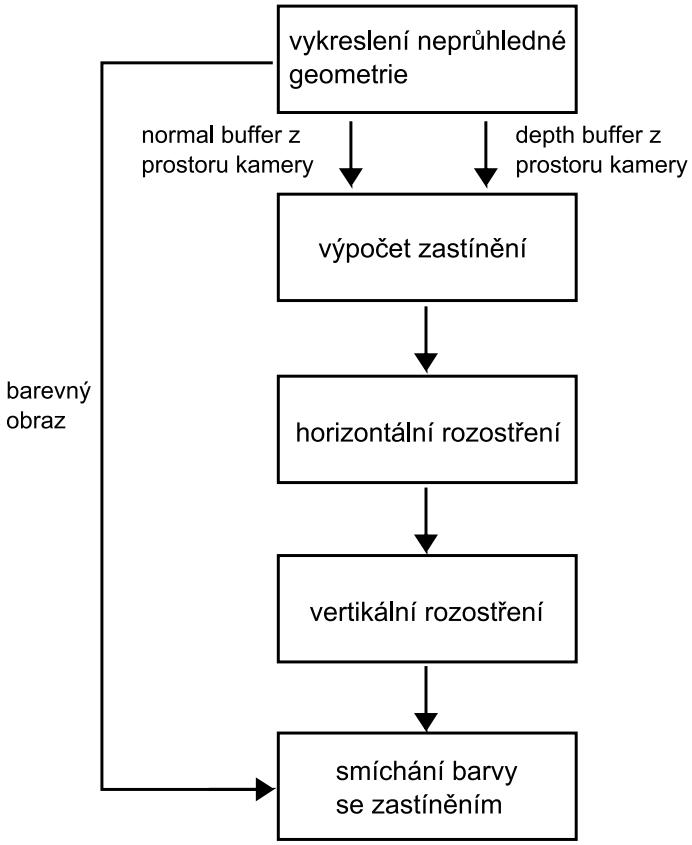
```
        ao += Falloff(d2) * (sin_s - sin_h);

        // nastaveni aktualniho horizontu
        tan_h = tan_s;
        sin_h = sin_s;
      }
    }
```

Výpis 10: Výpočet zastínění v určitém směru

4.3.4 Problém náhodných testovacích paprsků

Náhodná rotace testovacích směrů a jejich ofset má svůj význam. Zamezuje vytváření celistvých map podobného odstínu a opakování testovaných směrů vedoucích k nežádoucím artefaktům ve výsledném obraze zastínění. Na druhou stranu náhodné vzorky nevytváří plynulé přechody zastínění, které bychom potřebovali, ale vytváří v obraze šum. Aby byl výsledný dojem realističtější a nestahoval špatnou kvalitou pozornost, je třeba zastínění před smícháním s barevným obrazem rozmazat. Na obraz se použije gaussovské rozostření v horizontálním a vertikálním směru. Gaussovské rozostření lze vypočítat jedním průchodem obrazu, kdy se použije konvoluce s gaussovskou maskou. Pro urychlení výpočtu se využívá výhody gaussovského rozostření, kdy můžeme masku rozdělit, provést dva průchody (horizontální a vertikální) a minimalizovat tak množství potřebných výpočtů.

4.3.5 Vykreslovací smyčka



Obrázek 23: HBAO vykreslovací smyčka

V prvním kroku se vykreslí neprůhledná geometrie. Výstupem vykreslení je normal buffer a depth buffer obrazu z prostoru kamery a barevný obraz s phongovým stínováním. Textury normál a hloubek se využijí v dalším kroku při výpočtu zastínění. Dále následuje rozostření obrazu zastínění. Takto rozostřený obraz dále smícháme s barevným obrazem získaným v prvním kroku. Jelikož je obraz zastínění v odstínech šedi, můžeme provést míchání jako násobení barevného obrazu hodnotami obrazu zastínění.

4.3.6 Parametry zastínění

Velikost poloměru
Udává, jak velká bude testovací oblast.

Velikost bias úhlu
Velikost úhlu, který bude přičten k tečně bodu P pro odstranění nesprávného zastínění.

Počet směrů
Počet testovacích směrů kolem bodu P. Z této hodnoty se vypočte úhel, o který se budou jednotlivé směry paprsků otáčet kolem P.

Počet kroků
Počet testovaných horizontů v daném směru paprsku. Tímto číslem se vydělí velikost poloměru a získáme velikost jednoho testovacího kroku.

Kontrast
Udává míru zastínění. Touto hodnotou se vynásobí výsledná hodnota zastínění.

Velikost útlumu
Omezuje velikost útlumu zastínění. Hodnotou se upravuje útlum, kterým se násobí každá hodnota přispívající k zastínění.

Velikost rozostření
Upravuje rozptyl vzorků při výpočtu rozostření a udává tak míru rozostření.

4.3.7 Výsledky aplikace

Výsledné časy vykreslovací smyčky s HBAO algoritmem. Jedná se o průměrný čas každé úlohy na jeden snímek aplikace. Časy jsou měřené na mobilní grafické kartě NVIDIA GeForce GT 740M.

Počet směrů x počet kroků	Vygenerování depth bufferu	Výpočet zastínění	Výpočet barevného obrazu	Celkem	fps
4×4	3,08 ms	3,84 ms	3,11 ms	10,03 ms	83
8×8	3,08 ms	13,31 ms	3,08 ms	19,47 ms	44
16×16	3,22 ms	51,13 ms	3,07 ms	57,42 ms	17

Tabulka 4: Průměrné časy jednotlivých částí vykreslovací smyčky pro jeden snímek aplikace. HBAO algoritmus v rozlišení 800px $\times$ 600px.

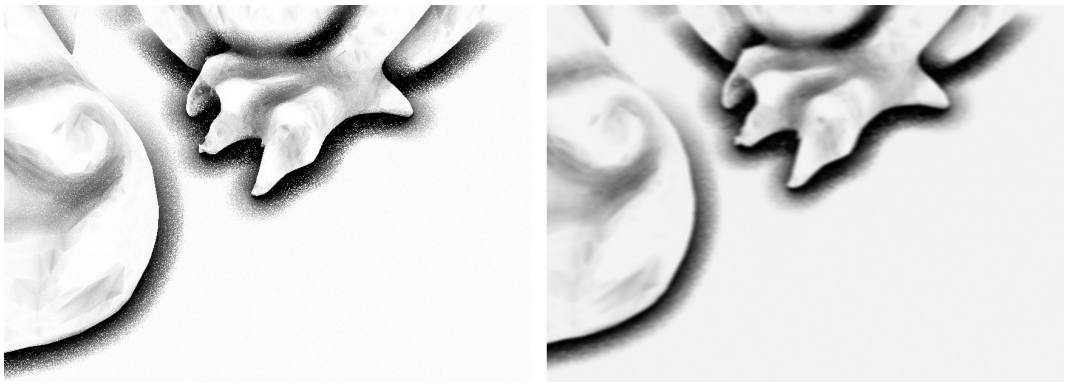
Počet směrů x počet kroků	Vygenerování depth bufferu	Výpočet zastínění	Výpočet barevného obrazu	Celkem	fps
4×4	3,58 ms	7,58 ms	3,38 ms	14,54 ms	55
8×8	3,57 ms	28,5 ms	3,42 ms	35,49 ms	26
16×16	3,58 ms	109,44 ms	3,45 ms	116,47 ms	8

Tabulka 5: Průměrné časy jednotlivých částí vykreslovací smyčky pro jeden snímek aplikace. HBAO algoritmus v rozlišení 1366px $\times$ 768px.

Počet směrů x počet kroků	Vygenerování depth bufferu	Výpočet zastínění	Výpočet barevného obrazu	Celkem	fps
4×4	5,1 ms	15,2 ms	4,27 ms	24,57 ms	33
8×8	5,0 ms	57,1 ms	4,26 ms	66,36 ms	14
16×16	4,66 ms	218,9 ms	4,26 ms	227,82 ms	4

Tabulka 6: Průměrné časy jednotlivých částí vykreslovací smyčky pro jeden snímek aplikace. HBAO algoritmus v rozlišení 1920px $\times$ 1080px.

Obrázek 24: Vykreslení části modelu Stanford dragon metodou HBAO s nastavením testovacích paprsků 4×4. Vlevo bez rozostření, vpravo s gaussovským rozostřením



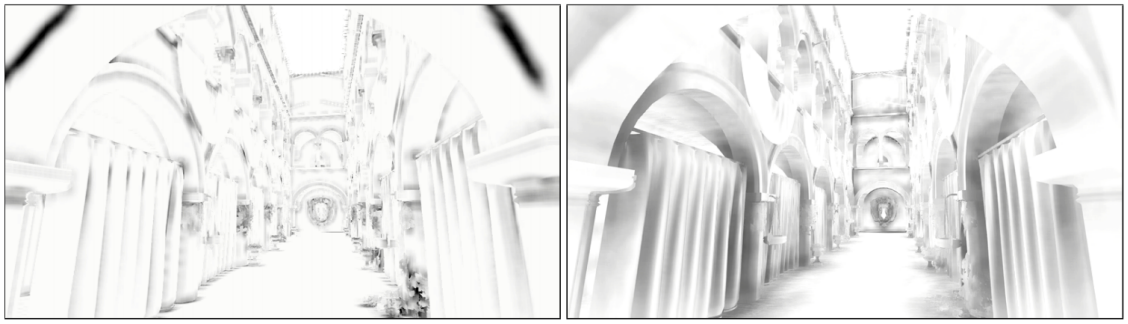
Obrázek 25: Vykreslení části modelu Stanford dragon metodou HBAO s nastavením testovacích paprsků 16×16. Vlevo bez rozostření, vpravo s gaussovským rozostřením

4.4 Scalable Ambient Obscurance

V roce 2011 přišlo herní studio Vicarious Visions s algoritmem pro řešení zastínění nazvaným Alchemy Ambient Obscurance (originál zde [8]). Pro výpočet je třeba mít depth

a normal buffer. Motivací autorů pro vytvoření nové metody zastínění bylo vytvoření přesnějšího a škálovatelného (přes velké prostory, až po vrásky v obličeji) řešení zastínění, které by mělo intuitivní parametry a bylo schopné realizovat zastínění na různých platformách (Xbox 360 až Windows s Direct3D 11 hardwarem) v rozmezí 3–5 ms.

Pro rozlišení 1280 $\times$ 720 a 12 vzorků na pixel trval výpočet na kartě GeForce 580 3 ms (4,5 ms na Xbox 360). Tyto výsledky by se zdály postačující a také získaly v herním průmyslu značnou pozornost. Nicméně McGuire, Mara a Luebke v roce 2012 vytvořili algoritmus založený na Alchemy AO, který je při zachování času výpočtu schopen vypočítat zastínění v rozlišení 1920 $\times$ 1080.



Alchemy AO [MOBH11]: $r = 0.10$m, 1280×720 in 2.3 ms **New SAO Result**: $r = 1.5$m @ 1920×1080 in 2.3 ms

Obrázek 26: Výsledek Scalable AO v porovnání s předchozím algoritmem Alchemy AO. Převzato z [8]

Scalable Ambient Obscurance odstraňuje potřebu normal bufferu a k výpočtu potřebuje znát pouze depth buffer. Ten je větší než velikost obrazovky, aby do zastínění mohly přispívat i části geometrie, které leží mimo viditelnou část scény. Nad těmito daty se provede několik výpočtů, určujících viditelnost bodu $0 \leq \mathrm{A} \leq 1$.

4.4.1 Přesnost depth bufferu

Jelikož se k výpočtu využívá pouze tohoto bufferu, je nutné aby obsahoval co nejpřesnější hodnoty. Vždy se pozornost soustředila na zvyšování přesnosti depth bufferu, ale autoři algoritmu se věnovali přesnosti rekonstrukce view-space pozic a normál ze znalosti depth bufferu. Protože každá aritmetická operace přináší chybu výpočtu, je třeba ji minimalizovat např. těmito kroky:

- výpočet modelview-projection provádět na CPU s přesností double, poté odeslat na GPU s přesností single
- volba vzdálené ořezové roviny $z_f = \infty$, minimalizující počet operací s plovoucí desetinnou čárkou
- při násobení vektoru se sloupcovou maticí používat násobení vektorem zleva $\mathbf{v} = \mathbf{v}^T\mathrm{P}$

4.4.2 Hierarchie depth bufferu

V tomto průchodu jsou hloubky převáděny z prostoru obrazu do prostoru kamery (pozorovatele). Po převodu je vytvořena hierarchie depth bufferu pro rychlejší výpočet zastínění. Pro převod každého pixelu depth bufferu z rozmezí $0 \leq \mathrm{d} \leq 1$ do $\mathrm{z} < 0$ je použita tato rovnice:

$$z(d) = \frac{c_0}{d \cdot c_1 + c_2} \ , \tag{18}$$

kde c = [z_n, -1, 1] pro $z_f = \infty$, jinak c = [$z_n z_f, z_n - z_f, z_f$]. Hodnoty z_n a z_f jsou vzdálenosti ořezových rovin projekce od pozice pozorovatele. Protože se budou hodnoty hloubek pixelů obrazu vyhledávat při vykreslování často, vytvoří se pro uchování informací o hloubce každého pixelu obrazu MIP mapy se snižujícím se rozlišením. Při procházení určité rovně MIP mapy zůstává její část v cache paměti a většina přístupů pro čtení bude z této paměti, což urychlí vyhledávání a výpočet.

4.4.3 Distribuce testovacích bodů

Tento krok vytváří s testovací bodů v hemisféře v prostoru kamery kolem bodu C. Pro získání testovacích bodů je třeba znát referenční bod C a jeho normálu n_C. Tyto hodnoty jsou vypočteny rekonstrukcí ze screen-space do view-space z hloubky pixelu obrazovky z_C = z(x′, y′) pomocí rovnic:

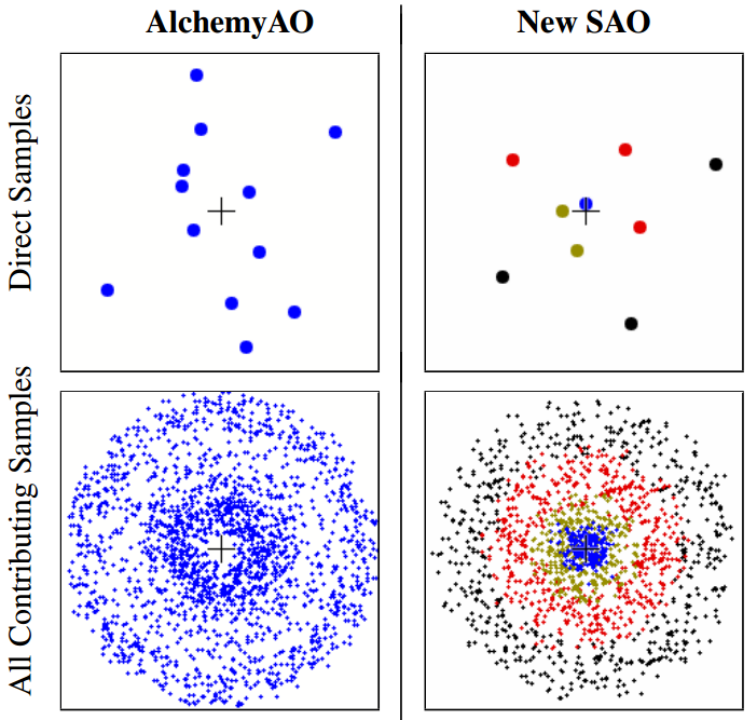
$$\begin{aligned} (x_C, y_C) &= z_C \cdot \left(\frac{1 - P_{0,2}}{P_{0,0}} - \frac{2(x' + \frac{1}{2})}{w \cdot P_{0,0}}, \frac{1 + P_{1,2}}{P_{1,1}} - \frac{-2(y' + \frac{1}{2})}{h \cdot P_{1,1}} \right) \\ \mathbf{n}_C &= normalize \left(\frac{\partial C}{\partial y'} \times \frac{\partial C}{\partial x'} \right) \ , \end{aligned} \tag{19}$$

kde w a h jsou rozměry obrazu, x′ a y′ jsou pozice pixelu obrazu. První rovnice získá pozici bodu v prostoru kamery pomocí inverzní projekce. Druhá rovnice získá normálu bodu C pomocí derivací ve směru x a y. Testovací vzorky jsou tvořeny odsazením, které se přičte k aktuální pozici pixelu (x′, y′). Vypočtené vzorky tvoří spirálu kolem tohoto bodu. Výpočet je následující:

$$\begin{aligned} \alpha_i &= \frac{1}{s}(i + 0.5) \\ r' &= \frac{-rS'}{z_c} \\ h_i &= r'\alpha_i \\ \theta_i &= 2\pi\alpha_i\tau + \phi \\ \mathbf{u}_i &= (\cos\theta_i, sin\theta_i) \ , \end{aligned} \tag{20}$$

kde r′ je poloměr testovacích vzorků, který odpovídá testované oblasti ve screen-sapce a je vztažený k vzdálenosti od pozorovatele. Parametr τ je počet rotací kolem bodu C, ϕ je

úhel náhodné rotace pixelu. Testovací bod vznikne přičtením odsazení k aktuální pozici pixelu obrazovky. Hodnota odsazení je h_i $\mathbf{u}_i$, kde $\mathbf{u}_i$ je směr posunu testovacího bodu od pozice (x', y') a h_i je vzdálenost, o kterou bude bod posunut. Díky kvalitnější distribuci testovacích vzorků a hierarchii depth bufferu je výpočet zastínění metodou SAO rychlejší než předchůdcem tohoto algoritmu - Alchemy AO.



Obrázek 27: Porovnání distribuce testovacích vzorků Alechemy AO a Scalable AO algoritmu. U SAO algoritmu jsou vyznačeny barevně jednotlivé MIP úrovně depth bufferu. Převzato z [8]

4.5 Multiview Ambient Occlusion

Tento algoritmus je založen na myšlence, že pro zdokonalení výsledného zastínění nestačí jen jeden pohled, ale je třeba zastínění počítat z více pozic ve scéně. Zároveň je použitelný pro prostředí se složitými geometriemi a velkým množstvím zastínění. Využívá znalosti depth bufferu z jiného pohedu než aktuálního, jako v případě, kdy ve scéně počítáme stíny. V tomto případě nepotřebujeme další zdroje pro výpočet zastínění, neboť využíváme znalosti depth bufferu scény z pohledu kamery a stínové mapy z pozice zdroje světla. Pohledů pro výpočet zastínění může být ve výsledku více, ale ve většině případů stačí k pohledu pozorovatele přidat pohled ze světelného zdroje (myšleno u venkovních scén). Výpočet zastínění se provádí libovolnou z předchozích metod. Algoritmus lze tedy použít na libovolnou screen-space metodu výpočtu zastínění. Informace o tomto algoritmu jsem čerpal z [9].

4.5.1 Váhování pohledů

U screen-space metod výpočtu zastínění se předpokládá viditelnost všech bodů obrazu, u kterých se počítá zastínění, a jejich okolí. Mohou nastat případy, kdy testované okolí bodu P obsahuje body, které by zastínění bodu mohlo nepříznivě ovlivnit. Proto tento algoritmus počítá s více pohledy a výsledné zastínění se vypočítá váhováním všech přispívajících pohledů. Pro n pohledů ve scéně je zastínění bodu P vypočteno jako:

$$O(\mathrm{P}) = \frac{\sum_{v=1}^{n} O_v(\mathrm{P}) w(v, \mathrm{P})}{\sum_{v=1}^{n} w(v, \mathrm{P})} , \tag{21}$$

kde v je aktuální pohled, O_v je zastínění v bodě pohledu v a $w(v, \mathrm{P})$ je váha v bodě P pro pohled v. Ke každému obrazu zastínění tak přibývá matice hodnot, která udává váhy každého pixelu pro výsledný obraz. Vliv konkrétního bodu P pohledu v na výsledný obraz ovlivňují tři faktory:

1. viditelnost bodu P v pohledu v
2. natočení plochy obsahující bod P vzhledem k pohledu v
3. vzdálenost projektovaného bodu P do pohledu v

Tyto vlastnosti ovlivňují velikost příspěvku zastínění určitého pohledu O_v(P) k celkovému zastínění. Velikost takového příspěvku pohledu v je dána váhovací funkcí jako:

$$w(v, \mathrm{P}) = b w_d(v, \mathrm{P}) + (1 - b) w_n(v, \mathrm{P}) , \tag{22}$$

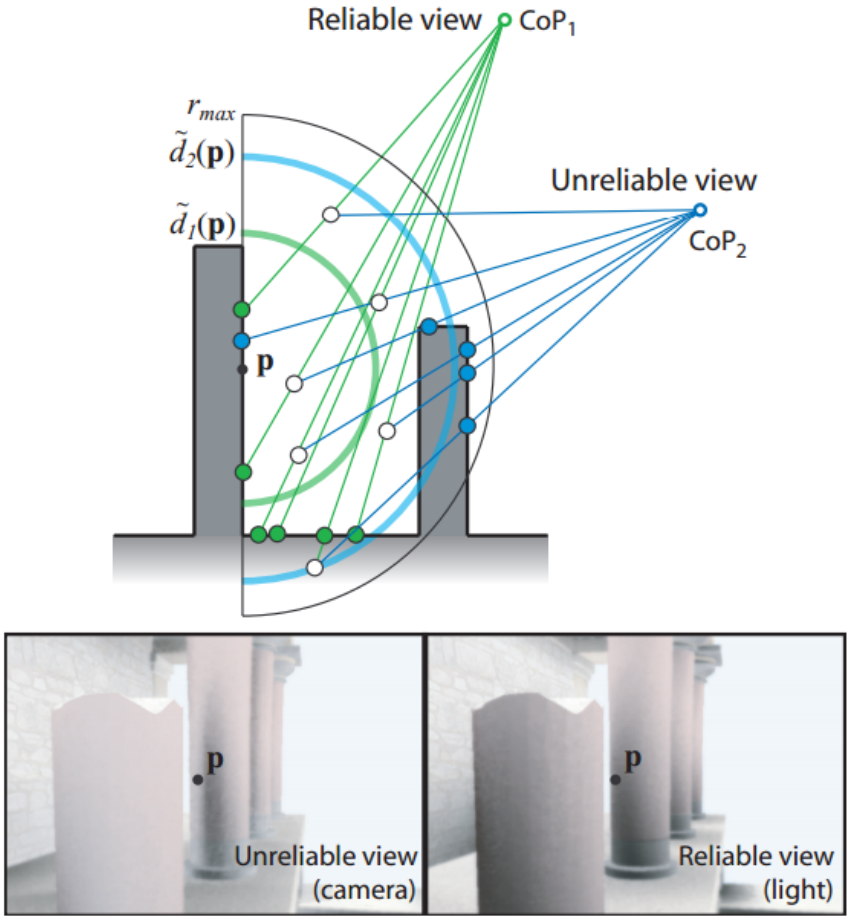
kde w_d (váha vzdálenosti) a w_n (váha směru) jsou váhovací funkce pro aktuální pohled v a b je koeficient smíchání těchto váhovacích funkcí. Pomocí tohoto koeficientu lze pozměnit výpočet zastínění tak, že je upravena priorita vlivu vzdálenosti bodů scény a vliv orientace těchto bodů na výsledný obraz.

4.5.2 Váha vzdálenosti

Váhovací funkce vzdálenosti ovlivňuje váhu bodu tak, že upřednostňuje projektované body do aktuálního pohledu, které jsou blíže aktuálnímu bodu P. Naopak vzdálenější body potlačuje. Zamezuje to počítání zastínění v místech skokových změn hloubek jako jsou okraje objektů, což by vedlo k nesprávným příspěvkům zastínění. Váha vzdálenosti pro bod P pohledu v je dána sledováním průměru vzdáleností testovacích vzorků od bodu P takto:

$$w_d(v, \mathrm{P}) = 1 - \frac{1}{N_s r_{\max}} \sum_{i=1}^{N_s} \min(\|\mathrm{P} - \mathrm{S_i}\|, r_{\max}) , \tag{23}$$

kde N_s udává počet testovacích vzorků a $r_{\max}$ je poloměr testovací hemisféry. Funkce min() zajišťuje testování uvnitř hemisféry a S_i je projektovaný bod do pohledu v. Výsledná váha udává nepřímo viditelnost bodu P v daném pohledu, neboť vyšší počet zastíněných bodů okolí zvyšuje průměrnou vzdálenost a snižuje váhu pohledu v.



Obrázek 28: MVAO váha vzdálenosti. Velké rozdíly mezi hloubkami v depth bufferu mohou vést ke vzniku nepravého zastínění. Tyto body je třeba potlačit. Převzato z [9]

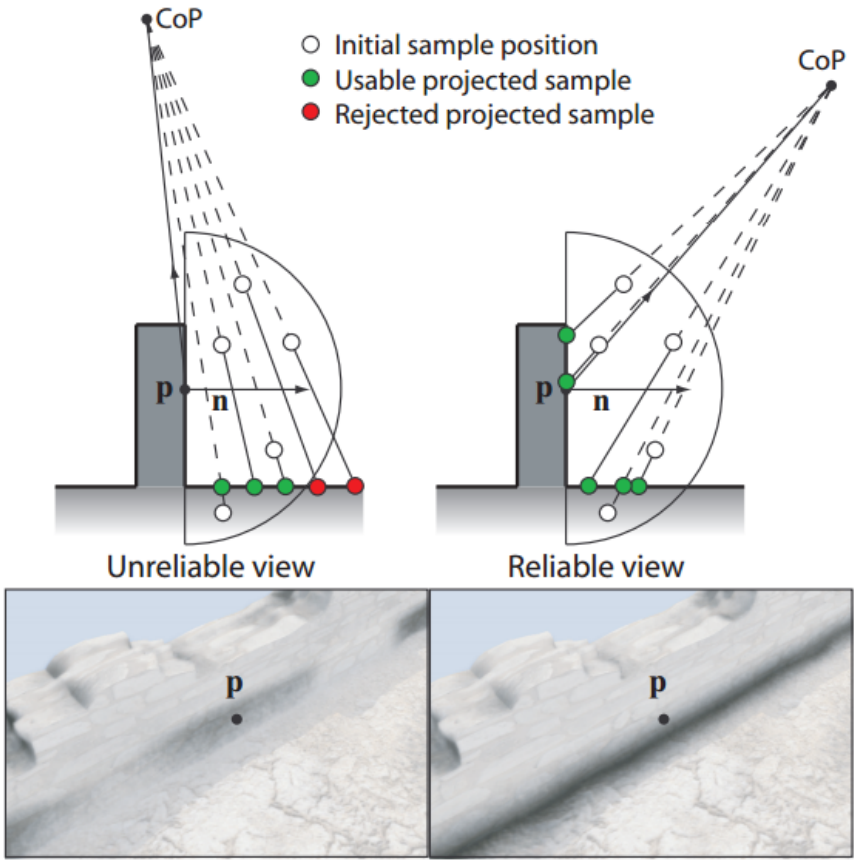
4.5.3 Váha směru

Funkce váhy směru snižuje vliv vzorků na povrchu, který směřuje stejným směrem jako aktuální pohled. V případě, kdy se normála povrchu se směrem pohledu rozbíhají, zvyšuje se pravděpodobnost projektování testovaného bodu na vzdálenou část geometrie nebo jiný objekt. Tyto vzorky by byly projektovány do pohledu pozorovatele s velkou pravděpodobností mimo oblast bodu P a způsobovaly by nechtěné zastínění. U zastínění

pomocí více pohledů je toto riziko podstatně větší. Proto je třeba těmto vzorkům snížit váhu, a tím je potlačit. Výpočet váhy směru je jednoduchý:

$$w_n(v, \mathrm{P}) = \max(0, \mathbf{l}_v \cdot \mathbf{n}) \ , \tag{24}$$

kde $\mathbf{l}_v$ je vektor směřující z bodu P do středu projekce pohledu v. Jedná se tedy o skalární součin normály povrchu $\mathbf{n}$ s tímto směrovým vektorem, který je omezen funkcí max() na maximální úhel 90° mezi těmito vektory.



Obrázek 29: MVAO váha směru. Projekce bodů směřujících od normály bodu P mohou způsobit zkreslení zastínění. Převzato z [9]

5 Závěr

Cílem této práce bylo porozumět a implementovat metody pro výpočet zastínění okolím, které jsou v posledních letech hojně využívány v herním průmyslu. Pro implementační část jsem zvolil dvě nejpoužívanější metody zastínění SSAO a HBAO. Ty jsem naimplementoval a otestoval na modelech z [13] a [14].

SSAO je starší metodou, která prošla vývojem a s omezením výpočtu na hemisféru kolem bodu podává použitelné výsledky. Zvolil jsem si ji proto, že je jednoduchá na implementaci a je nejpoužívanější metodou výpočtu zastínění v reálném čase. Není sice tak přesná, ale je rychlejší na výpočet. Nakonec jsem byl jejími výsledky překvapen.

HBAO je druhá metoda kterou jsem zvolil pro implementaci, neboť je novější a je neustále vyvíjena a optimalizována pro rychlejší vykreslování, aby postupně nahradila SSAO. Díky hledání horizontu v depth bufferu jsem byl schopen docílit přesnějších výpočtů tam, kde postupně dochází ke změnám hloubky v testovaném směru. Algoritmus je schopen rozpoznávat změny hloubky po cestě paprsku a tím dává vyniknout detailům objektu.

Rychlost obou algoritmů jsem testoval na svém notebooku s grafickou kartou NVIDIA GeForce GT 740M. Pro maximální testované rozlišení obrazu (1920×1080) trvalo vygenerování textur (s hloubkami a normálami pixelů), potřebných pro výpočet zastínění, a samotný výpočet zastínění metodou SSAO se 48 testovacími vzorky pouhých 25,78 ms. U HBAO algoritmu s podobným počtem vzorků (8×8) tento výpočet trval 62,1 ms. To je více než dvojnásobný nárůst výpočetního času, který favorizuje SSAO metodu. HBAO je modulárnější metoda, u které lze nastavovat více parametrů (např. bias, útlum), díky kterým vytváří v určitých případech přesvědčivější výsledky. Tyto parametry by bylo možné přidat i do SSAO, avšak s dopadem na výkon. Myslím si tedy, že v mnohých případech postačuje jednodušší řešení zastínění pomocí SSAO.

Při implementaci těchto algoritmů jsem využil znalostí z grafických předmětů u problémů s transformacemi mezi prostory, změnami souřadných systémů a vztahů mezi vektory v prostoru. Pro implementaci těchto metod jsem prostudoval algoritmy řešící globální osvětlení, které mě zaujaly a rád bych se o tuto oblast v budoucnu zajímal.

6 Reference

[1] Pharr, M., Fernando, R. Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation. 2005. Addison-Wesley Professional.

[2] Shirley, P., Morley, R. K. Realistic ray tracing. AK Peters, Ltd., 2003.

[3] Sojka, E. Počítačová grafika II: Metody a nástroje zobrazování 3D scén. 1 Ostrava: VŠB TUO, RCCV. 2003. ISBN 80-248-0293-7

[4] Jensen, Henrik W., Realistic Image Synthesis Using Photon Mapping, A K Peters, Ltd., Massachusetts, 2001.

[5] Mittring, M. Finding Next Gen - CryEngine 2. 2007. Dostupné z URL: <http://developer.amd.com/wordpress/media/2012/10/Chapter8-Mittring-Finding_NextGen_CryEngine2.pdf>.

[6] Dimitrov, R., Bavoil, L., Sainz, M. Horizon-Split Ambient Occlusion. 2007. Dostupné z URL: <http://www.twistedsanity.com/rdimitrov/Horizon-SplitAmbientOcclusion(2007).pdf>.

[7] Bavoil, L., Sainz, M. Image-Space Horizon-Based Ambient Occlusion. 2008. Dostupné z URL: <http://developer.download.nvidia.com/presentations/2008/SIGGRAPH/HBAO_SIG08b.pdf>.

[8] McGuire, M., Mara, M., Luebke, D. Scalable Ambient Obscurance. 2012. Dostupné z URL: <http://graphics.cs.williams.edu/papers/SAOHPG12/McGuire12SAO.pdf>.

[9] Vardis, K., Papaioannou, G., Gaitatzes, A. Multi-view Ambient Occlusion with Importance Sampling. 2013. Dostupné z URL: <http://graphics.cs.aueb.gr/graphics/docs/papers/MultiviewAmbientOcclusion.pdf>.

[10] Pharr, M., Green, S. Ambient Occlusion. 2004. Dostupné z URL: <http://http.developer.nvidia.com/GPUGems/gpugems_ch17.html>.

[11] Zhukov, S., Iones, A., Kronin, G. An ambient light illumination model. In Rendering Techniques' 98 (pp. 45-55). Springer Vienna.

[12] Rendering equation. Dostupné z URL: <http://en.wikipedia.org/wiki/Rendering_equation>. Poslední úpravy 10. 9. 2013.

[13] Levoy, M., Gerth, J., Curless, B., Pull, K. The Stanford 3D scanning repository. 2005. Dostupné z URL: <http://www.graphics.stanford.edu/data/3Dscanrep/>.

[14] McGuire, M. Computer Graphics Archive. 2011. Dostupné z URL: <http://graphics.cs.williams.edu/data>.

A Ukázka SSAO na modelu Stanford dragon

Obrázek 30: Model Stanford dragon vykreslen zastíněním SSAO se 48 testovacími paprsky

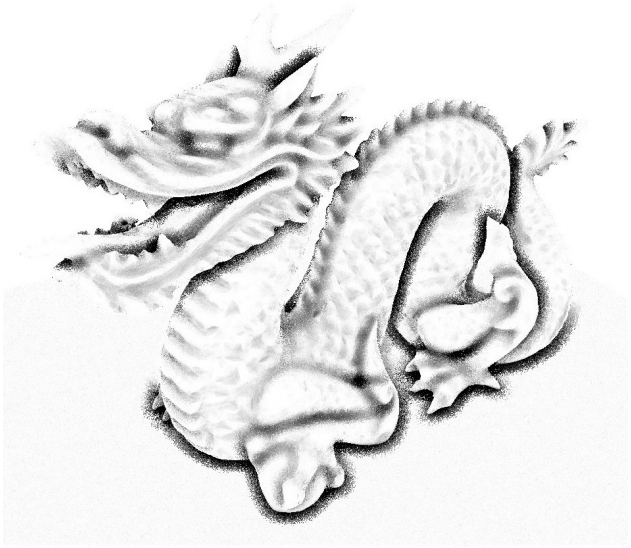
Obrázek 31: Model Stanford dragon vykreslen opět zastíněním SSAO se 48 testovacími paprsky. Nyní s rozostřením, které zakrývá šum v obraze

B Ukázka SSAO na modelu Sponza

Obrázek 32: Model Sponza vykreslen zastíněním SSAO se 48 testovacími paprsky. Je vidět šum a chybné zastínění na hranách uvnitř oblouků

Obrázek 33: Model Sponza vykreslen opět zastíněním SSAO se 48 testovacími paprsky a rozostřením efektu zastínění zakrývající šum

C Ukázka HBAO na modelu Stanford dragon



Obrázek 34: Model Stanford dragon vykreslen zastíněním HBAO s 8 testovacími směry a 8 kroky

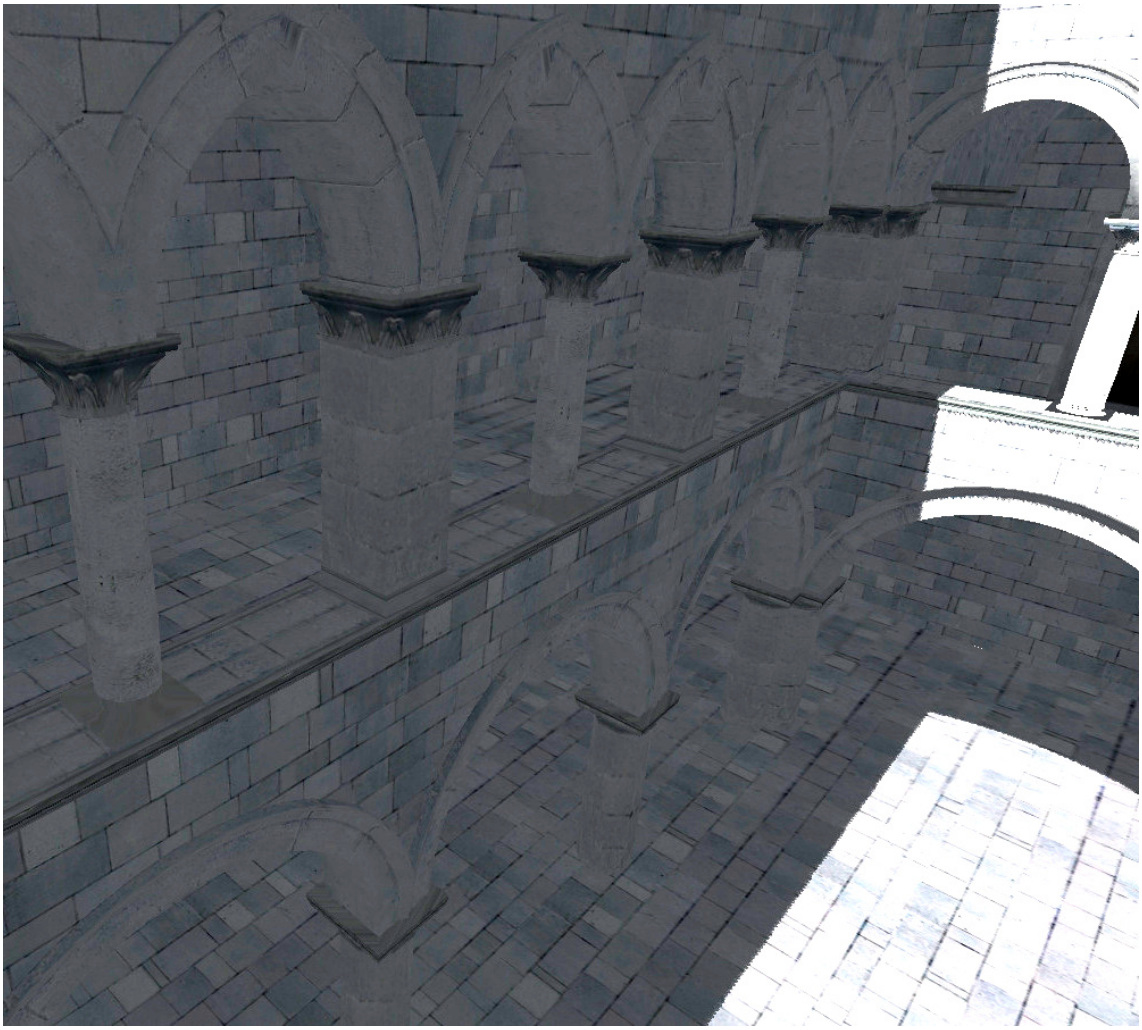
Obrázek 35: Model Stanford dragon vykreslen zastíněním HBAO s 8 testovacími směry, 8 kroky a rozostřením

D Ukázka HBAO na modelu Sponza

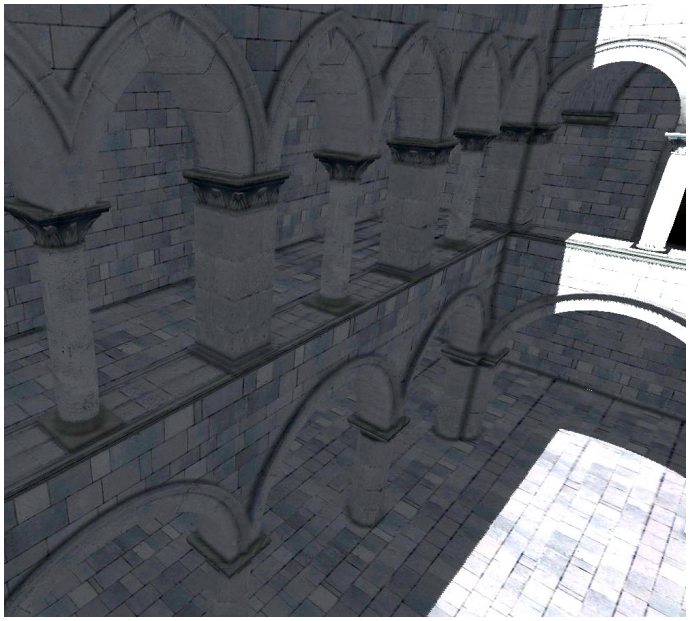
Obrázek 36: Model Sponza vykreslen zastíněním HBAO s 8 testovacími směry, 8 kroky a úhlem biasu 10 stupňů

Obrázek 37: Model Sponza vykreslen zastíněním HBAO s 8 testovacími směry, 8 kroky. Opět přidán bias 10 stupňů a rozostření

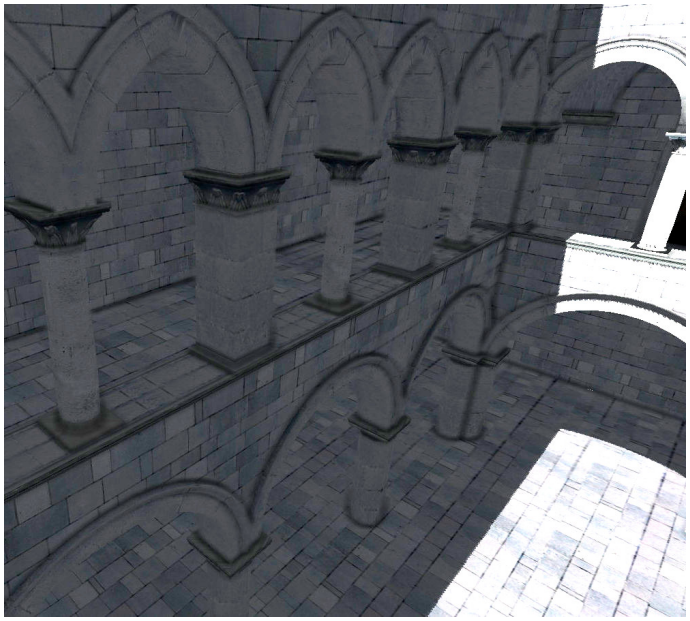
E Barevný obraz modelu Sponza



Obrázek 38: Barevný obraz modelu Sponza bez zastínění

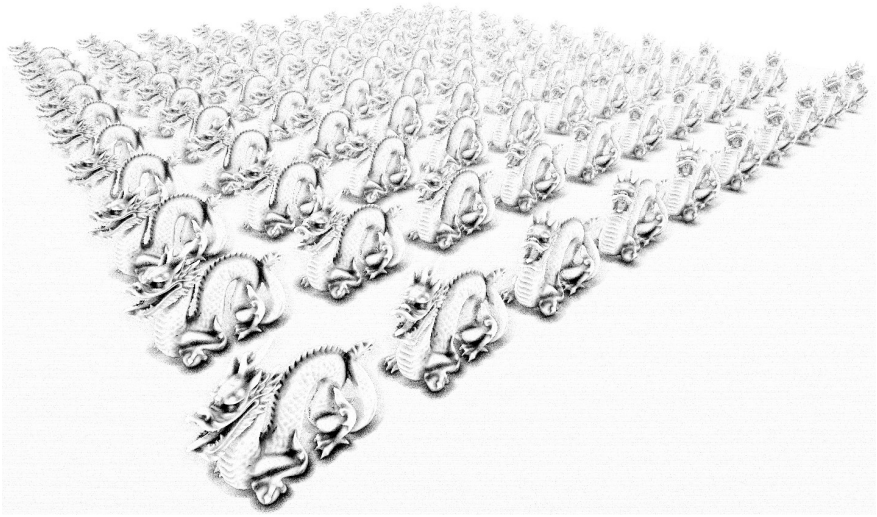


Obrázek 39: Barevný obraz modelu Sponza s rozostřeným zastíněním SSAO

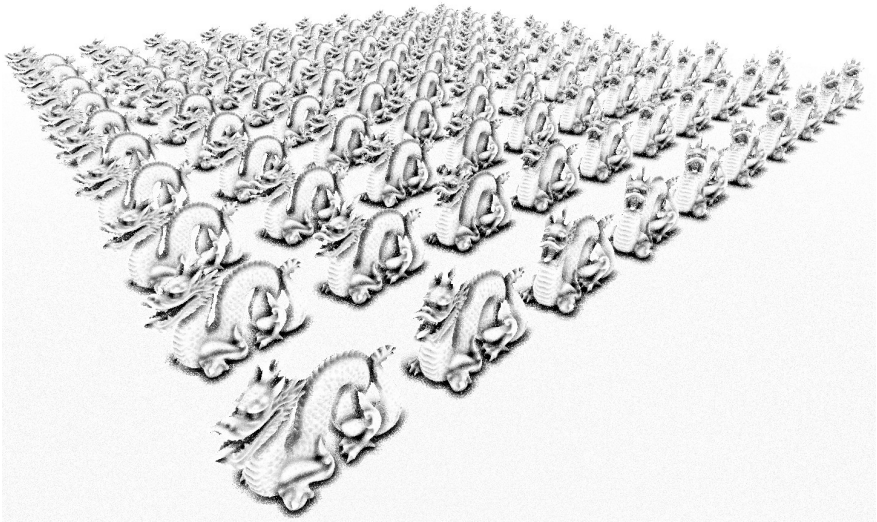


Obrázek 40: Barevný obraz modelu Sponza s rozostřeným zastíněním HBAO. U HBAO metody je vidět zastínění i v místech geometrie, která je natočená od pozorovatele (oblouky, sloupy)

F Vykreslení 100 modelů Stanford dragon



Obrázek 41: Vykreslení 10M trojúhelníků metodou zastínění SSAO se 32 testovacími paprsky. Vykreslení scény (vygenerování depth a normal bufferu) trvalo 104 ms a výpočet zastínění 15 ms



Obrázek 42: Vykreslení 10M trojúhelníků metodou zastínění HBAO se 6 testovacími směry a 6 kroky v každém směru. Vykreslení scény (vygenerování depth bufferu) trvalo 102 ms a výpočet zastínění 12 ms

G Vykreslení 1024 modelů Stanford dragon

Obrázek 43: Vykreslení 102M trojúhelníků metodou zastínění SSAO se 32 testovacími paprsky. Vykreslení scény (vygenerování depth a normal bufferu) trvalo 1003 ms a výpočet zastínění 15 ms

Obrázek 44: Vykreslení 102M trojúhelníků metodou zastínění HBAO se 6 testovacími směry a 6 kroky v každém směru. Vykreslení scény (vygenerování depth bufferu) trvalo 1002 ms a výpočet zastínění 14 ms